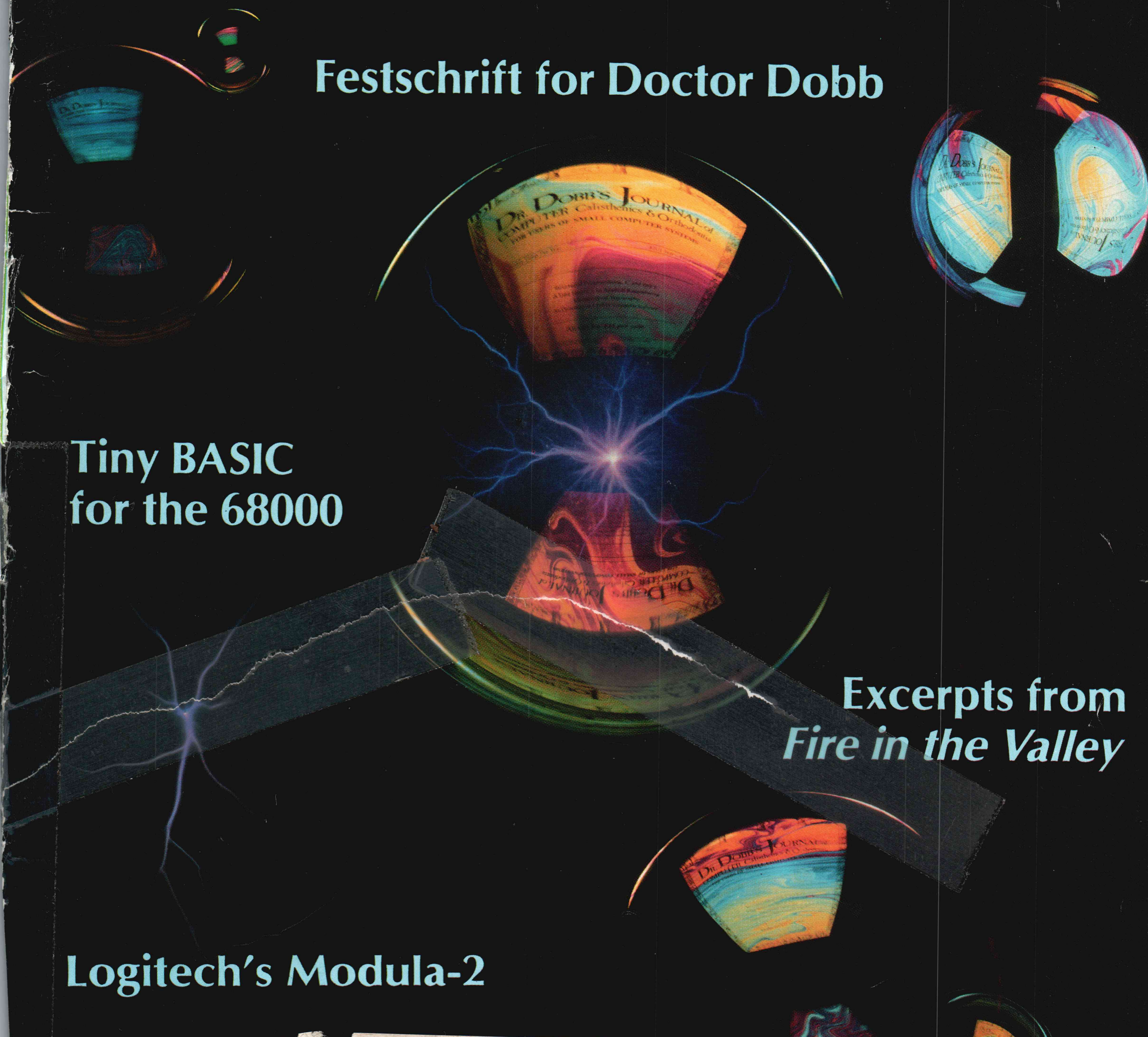
100th Issue!

SOFTWARE TOOLS FOR ADVANCED PROGRAMMERS

Dr. Dobb's Journal

#100 February 1985

$2.95 (3.50 Canada)



Festschrift for Doctor Dobb

Tiny BASIC for the 68000

Excerpts from *Fire in the Valley*

Logitech's Modula-2

TURBO TOOLBOX
TURBO PASCAL
Turbo Tutor
A Turbo Pascal Tutorial
BORLAND INTERNATIONAL

Imagine dBASE III™ running up to 20 times faster.

The time for Clipper has arrived.

Clipper introduces you to the time of your life.

Time is your most valuable commodity. Because how you spend your time, is how you live your life.

At Nantucket, we believe you should live life to the fullest.

Clipper, the first true compiler for dBASE III,™ is a timely example. Now, dBASE compiled by Clipper runs 2 to 20 times faster than dBASE with its standard interpreter.

A dBASE interpreter painstakingly checks and executes your source code one line at a time, every time you run a program. With Clipper, once you've debugged your source code, it's compiled into more efficient machine code. Your program runs without the time-consuming overhead of redundant translation. Clipper compiles all your existing and future dBASE III programs.

Developing a compiler for dBASE III was just a matter of time. Call your dealer or our toll free 800 number and ask for Clipper.

Then go make the most of your life time.

Nantucket

20456 Pacific Coast Hwy., Malibu, Ca. 90265 (800) 556-1234 ext. 225. In California (800) 441-2345 ext. 225

dBASE III is a registered trademark of Ashton-Tate

Circle **no. 56** on reader service card.

Dr. Dobb's Journal

Editorial

Editor-in-Chief *Michael Swaine*
Managing Editor *Randy Sutherland*
Assistant Editor *Frank DeRose*
Technical Editor *Alex Ragen*
Contributing Editors *Robert Blum, Dave Cortesi, Ray Duncan, Anthony Skjellum, Michael Wiesenberg*
Copy Editors *Polly Koch, Cindy Martin*
Typesetter *Jean Aring*
Editorial Intern *Mark Johnson*

Production

Design/Production Director *Detta Penna*
Art Director *Shelley Rae Doeden*
Production Assistant *Alida Hinton*
Cover *Tom Upton*

Advertising

Advertising Director *Stephen Friedman*
Advertising Sales *Walter Andrzejewski, Shawn Horst, Beth Dudas*
Advertising Coordinators *Alison Milne, Lisa Boudreau*

Circulation

Circulation and Promotions Director *Beatrice Blatteis*
Fulfillment Manager *Stephanie Barber*
Direct Response Coordinator *Maureen Snee*
Promotions Coordinator *Jane Sharninghouse*
Circulation Assistant *Kathleen Boyd*

M&T Publishing, Inc.

Chairman of the Board *Otmar Weber*
Director *C.F. von Quadt*
President *Laird Foshay*

Dr. Dobb's Journal (USPS 307690) is published monthly by M&T Publishing, Inc., 2464 Embarcadero Way, Palo Alto, CA 94303, (415) 424-0600. Second class postage paid at Palo Alto and at additional entry points.

Address correction requested. Postmaster: Send Form 3579 to *Dr. Dobb's Journal*, 2464 Embarcadero Way, Palo Alto, CA 94303. **ISSN 0278-6508**

Subscription Rates: $25 per year within the United States, $44 for first class to Canada and Mexico, $62 for airmail to other countries. Payment must be in U.S. Dollars, drawn on a U.S. Bank.

Contributing Subscribers: Christine Bell, W.D. Rausch, DeWitt S. Brown, Burks A. Smith, Robert C. Luckey, Transdata Corp., Mark Ketter, Friden Mailing Equipment, Frank Lawyer, Rodney Black, Kenneth Drexler, Real Paquin, Ed Malin, John Saylor Jr., Ted A. Reuss III, InfoWorld, Stan Veit, Western Material Control, S.P. Kennedy, John Hatch, Richard Jorgensen, John Boak, Bill Spees, R.B. Sutton. **Lifetime Subscribers:** Michael S. Zick, F. Kirk.

Foreign Distributors: ASCII Publishing, Inc. (Japan), Computer Services (Australia), Computer Store (New Zealand), Computercollectief (Nederland), Homecomputer Vertriebs GMBH (West Germany), International Presse (West Germany), La Nacelle Bookstore (France), McGill's News Agency PTY LTD (Australia), Progresco (France).

People's Computer Company

Dr. Dobb's Journal is published by M&T Publishing, Inc. under license from People's Computer Company, 2682 Bishop Dr., Suite 107, San Ramon, CA 94583, a non-profit, educational corporation.

February 1985
Volume 10, Issue 2

CONTENTS

In This Issue

For this 100th issue of *DDJ,* Mike Swaine interviewed the men whose names became the Doctor's: Dennis Allison and Bob Albrecht. Dennis is in "Software Designer" and Bob is in "Festschrift." To meet Bob and Dennis is to understand why the "temporary" project they started almost a decade ago is still gathering momentum. Bob and Dennis are visionary, and powerful. Suzanne Rodriguez' contribution to the "Festchrift" provides an interesting glimpse of the way things were at *People's Computer Company* under Bob and Dennis' leadership. Excerpts from *Fire in the Valley* provide a wider context for what was happening. We couldn't resist dropping in Gordon Brandly's "Tiny BASIC for the 68000" to link our roots with the state of the art.

The inclusion of Charles Burton's "An Enhanced ADFGVX Cipher System" reflects a philosophy that has continued from an early *People's Computer Company* poster: "Use computers for people, not against them." The power to encrypt one's communications, once the domain of high government and the military, has now been placed in the hands of the people. Since 1976, we have enjoyed taking what is secret, hidden, expensive, protected, or otherwise inaccessible, and placing it in the public domain.

This Months Cover

This months cover was created by Tom Upton. The photograph shows bubbles with sparks reflecting and old *DDJ* cover.

Next Month

The March issue will announce the winner(s) of the AI programming competition. The issue is shaping up to be quite Prolog oriented. We would really like to work up a public domain Prolog for micros. Contributions? In the "Realizable Fantasies" column next month you will read of a proposed project to write a complete public domain Unix, called GNU (acronym for "GNU's Not Unix"). In the same vein, the author of the grep.c program that we published in October 1984, Allen Holub, will pick up where Anthony Skjellum left off and become our new C columnist. We are pleased to announce that the C column will now run every month.

This Month's Referees

Dave Cortesi, Resident Intern — John Taber, IBM

SOFTWARE TOOLS FOR ADVANCED PROGRAMMERS
Dr. Dobb's Journal

ARTICLES

Festschrift for Doctor Dobb 26 Some old friends flame about the past, present and future in honor of the Doctor's 100th issue **(Reader Ballot No. 192)**
Contributions by Suzanne Rodriguez, Tom Pittman and Bob Albrecht

Fire in the Valley 32 The Homebrew Computer Club, Processor Technology, the S100 bus meetings **(Reader Ballot No. 193)**
by Michael Swaine and Paul Freiberger

Tiny BASIC for the 68000 42 Li Chen Wang's original Palo Alto Tiny BASIC, now in 68000 code, with installation instructions and a bulletin board number that carries the code **(Reader Ballot No. 194)**
by Gordon Brandly

An Enhanced ADFGVX Cipher System 48 An enhanced version of the ADFGVX cipher system, written in C, wth a brief history of the original German system **(Reader Ballot No. 195)**
by C.E. Burton

More dBASE Tips and Techniques 71 A program FLIPIT that changes the command file extension .CMD to .PRG and vice versa **(Reader Ballot No. 199)**
by Gene Head

COLUMNS

Dr. Dobb's Clinic 16 Throughput from various perspectives **(Reader Ballot No. 190)**
by D.E. Cortesi

C/Unix Programmer's Notebook 22 C style, comments on printf() **(Reader Ballot No. 191)**
by Anthony Skjellum

Software Designer 118 Running light in 1985 **(Reader Ballot No. 196)**
by Michael Swaine

CP/M Exchange 96 Buffered disk I/O **(Reader Ballot No. 197)**
by Bob Blum

DEPARTMENTS

Editorial 6

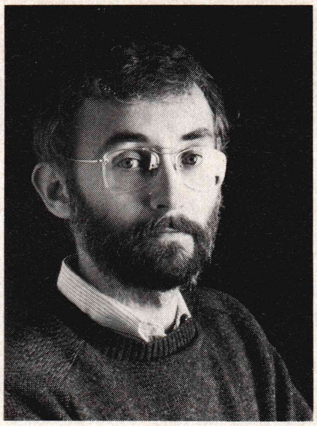
Letters 8

Reviews 74 Logitech's Modula-2

Books 120

Of Interest 124 **(Reader Ballot No. 198)**

Advertiser Index 128



Before they put me on the payroll here, I was a loyal reader of *DDJ*. This is my personal thank-you to some of the people who created *DDJ* over the past hundred issues. I've left out many people through oversight, but here are well over a hundred.

Thank you,

Phyllis Adams, Bob Albrecht, Dennis Allison, Robin Allison, Walter Andrzejewski, Julie Anton, Jean Aring, John Arnold,
Rick Bakalinsky, Stephanie Barber, Jeannie Barroga, Sam Bassett, Anatta Blackmarr, Beatrice Blatteis, Robert Blum, Christine Botelho, Lisa Boudreau, Kathleen Boyd, Sally Brenton, Bill Bruneau,
Ron Cain, Dave Caulkins, Ward Christensen, Maureen Christine, Peter Clark, Dave Cortesi, Gavin Cullen, Carole Cullenbine,
Leah Dansby, Jim Day, Frank DeRose, Shelley Rae Doeden, Steve Dompier, Beth Dudas, Ray Duncan,
Gordon Eubanks,
Paula Fairchild, Fred Fehlau, Lee Felsenstein, Eugene Fisher, Laird Foshay, Stephen Friedman,
Mike Gabrielson, Bill Gale, Gary Gaugler, Mag Glick, Greg Gordon, H.T. Gordon, F.J. Greeb, Richard Grigonis, Ed Gueble,
Aleeca Hamilton, Craig Harper, David C. Harris, Kim Harris, Hank Harrison, Gene Head, Michael Heatherman, Matthew Heiler, Jim Hendrix, Andy Hertzfeld, Nancy Heubach, Alice Hinton, Susan Hinton, Tita Hinton, Willard Holden, Allen Holub, Shawn Horst, Alana Hunter,
Meredith Ittner,
Bob Jacobsen, John S. James, Mark Johnson, Do-While Jones,
Laura Kares, Maria Kent, Gary Kildall, Art Kleiner, Polly Koch, Cynthia Kosina, Craig LaGrow, Jane Laidley, Carl Landeau, Fritzi Lareau, J.L. Lawrence,
Michael Madaj, Rosehips Malloy, Linda Marohn, Cindy Martin, Tom Martin, Terri Marty, W.D. Maurer, Dwight McCabe, Al McCahon, Dennis McGhie, Jane McKean, Lorraine McLaughlin, Mary Jo McPhee, Laura Mendell, Ann Merchenberger, Doug Millison, Alison Milne, Ann Miya, Michelle Moonstone, Carl Moser, Erik Mueller, Andrea Nasher,
Theresa O'Rourke, Marlin Ouverson,
Sahnta Pannuti, Ronald Parsons, Janet Payne, Detta Penna, Giuliana Peter, Tom Pittman, Terri Pond, Beverly Pyle,
Bill Ragsdale, Jef Raskin, Ed Ream, Pete Roberts, Beverly Robinson, Suzanne Rodriguez, Betsy Roeth, Gloria Romanoff, Dan Rosset, Barbara Ruzgerian, Barbara Rymsa,
Jonathan Sachs, Kent Safford, Carol Sevilla, Jane Sharninghouse, Anthony Skjellum, Maureen Snee, John Starkweather, Susan Strange, Robert Suding, Randy Sutherland,
Ari Taylor, Frank Trujillo, Marianne Tryens, Margaret Tsai,
Lichen Wang, Jim Warren, Judith Wasserman, Sara Werry, Ann West, Clifford West, Charles Wetherell, Dick Whipple, Michael Wiesenberg, Renny Wiggins, Tom Williams, Steve Willoughby, Marvin Wizenread, Donna Lee Wood, Steve Wozniak,
Greg Yob
and Michael Zick.

Michael Swaine

Michael Swaine

Minimax Exchange Algorithm

Dear Sir:
I was very interested in the Minimax algorithm presented by Steven A. Ruzinsky in *DDJ* No. 93 (July 1984). He presented several approximations that were better than the generally published series. Seeing this, I was inspired to program a form of the exchange algorithm and compare my results with his. Given the work that obviously went into his approach, I was surprised that even his results could be improved upon. In particular, for *sin (pi*x/2)* approximated by a 5-term series, Ruzinsky gave a maximum relative error of 5.31748E-09. My exchange algorithm gave a maximum relative error of 5.31399E-09. While this difference is so small as to be almost trivial, I believe it points up a logical error in Ruzinsky's approach.

To understand this error one must first realize that for an L_∞ approximation with N parameters, the approximation is guaranteed to reach its maximum error at least N + 1 times. Ruzinsky's algorithm tries to find these N + 1 or more places by giving ever-increasing weights to predetermined points near the actual maximum error positions, but, of course, these points are virtually never in the right positions for a true L_∞ approximation.

The exchange method, on the other hand, is directed toward finding these N + 1 points and building an approximation around them. When it works, it works best, but it appears to be far more finicky than Ruzinsky's algorithm.

In fact, Ruzinsky actually underestimates the maximum error of his approximations because the maximum error of his approximations does not occur at his lattice points, but somewhere in between.

In light of all this I propose the following modification of Ruzinsky's algorithm: after the Speedup step, which halves the number of lattice points, add new lattice points midway between each of the remaining points. This has the effect of increasing the lattice density near the maximum error and, I expect, will give the correct L_∞ approximation.

A few technical notes: GW BASIC on the Eagle PC that I used, like most Microsoft BASIC interpreters, does not support double precision transcendentals, so my first order of business was implementing *sin(p*x/2)* in double precision. This is not too bad if you don't care much about speed. Also the coefficients I got for Figure 8, page 94, are:

```
A(0) =  1.570796318447697
A(1) = -0.6459637105998668
A(2) =  0.0796896789479709
A(3) = -4.67376661266352E-03
A(4) =  1.514851308552791E-04
```

I also tried *arctan(x)* and again found a small improvement on Ruzinsky's result.

Finally, I would be interested in how Mr. Ruzinsky linearized his various functions. Perhaps the contest he mentioned (for a free one-year subscription to *DDJ*) could be ended.

Sincerely,
Allen E. Tracht
12469 Cedar Rd., #11
Cleveland Heights, OH
44106

Forth Standards

Dear Editor:
As a Forth vendor, I must take issue with your September 1984 article by Ray Duncan and Martin Tracy, "The FVG Standard Floating-Point Extension." I would like to point out that the document discussed in the article was based on the input of about eight vendors (including the authors). Most of these vendors do not produce floating-point applications systems. In fact, those of us who use the pre-existing Mountain View Press Floating-Point Standard were quite shocked to see this material in print. My own company has a mature floating-point system with utilities, matrix math, and FFTs, which has been available for over two years and was purchased by Martin Tracy of Micromotion about a year ago. This new "standard" bears little resemblance to the working standard that has come out of the experience of several years of real applications programming with floating-point Forth by those at ForthKit, Redshift Limited, and Mountain View Press.

In a recent editorial [October 1984] you commented that the FVG Standard showed that Forth was maturing and was no longer a language in flux. This conclusion is certainly inaccurate.

Micromotion and Creative Solutions (among others) were invited over a year ago to participate in a formal standards effort sponsored by Mountain View Press. They declined to participate. Through a great deal of work, those companies that did contribute developed a solid, mature standard. The programming systems based on our standard have received a great deal of use in laboratories and schools around the world; we consider them to have passed the test of time.

Something also needs to be said about the use of separate floating-point stacks. The FVG document claims that applications adhering to the names and functions of the "standard" will be transportable among the various floating-point implementations. Unfortunately, this is not the case. The paragraph on floating-point stacks says that the document does not concern itself with the issue of separate stacks. The

document states that on systems with a separate floating-point stack the source or destination of a real number argument is the floating-point stack. These statements seem to ignore the fact that programs written for a separate stack will not transport to a system that uses the parameter stack for floating-point numbers and vice versa.

A separate floating-point stack is necessary for efficient coding of complex algorithms. I used a parameter stack-based system for a year before switching to a separate floating-point stack in 1981, and I was amazed to see all my code simplify. But converting from one system to the other is a tedious process and involves rethinking and rewriting all the code—this can hardly be called transportable! The Mountain View Press Standard specifies a separate stack to aid transportability, although it is our feeling that floating-point applications cannot be absolutely portable, due to the differences in hardware and software number packages.

I feel that Ray Duncan has performed a disservice to the Forth community by publishing this confusing document and further clouding the floating-point standards issue.

Sincerely,
Charlie Springer
Redshift Limited
417 Forest Ave.
Palo Alto, CA 94301

Forth Compiler Changes

Dear Editor,
In my article "A Forth Native-Code Cross Compiler for the MC68000" (September 84) there are a couple of errors that should be corrected. I have also done some additional work that should make the compiler more portable.

On page 71 (second column, first paragraph) the sentence:

> Each time 2* is called, the code implementing it is *sorted* in the host dictionary, and the dictionary pointer is updated.

should read:

> Each time 2* is called, the code implementing it is *stored* in the

host dictionary, and the dictionary pointer is updated.

On page 75 under "Final Comments" the sentence:

> This word should reset all of the compiler variables in screen 9 to their original *style* and generate an appropriate header to permit the operating system to load and execute the module.

should read:

> This word should reset all of the compiler variables in screen 9 to their original *values* and generate an appropriate header to permit the operating system to load and execute the module.

I have used the compiler on a Forth-83 system (F83 model by Laxen and Perry) with very few changes. The following is a list of the changes that need to be made:

1. Change the variable initialization in screens 9 and 10 to conform to the Forth-83 standard.
2. Change the definition of the word HIGH-BYTE on screen 12 to the following.
   : HIGH-BYTE FLIP ; (FLIP is not Forth-83 standard, for F83 only.)
3. Change the word <BUILDS on screens 14 and 17 to CREATE or use the definition of <BUILDS given in screen 44 line 10.
4. Delete the word HERE from the definition of BYTES on screen 21.
5. Delete the word ;S from screens 24, 33, and 43. (Not all systems support this word and it is not required.)
6. Change the phrase A BYTES to 0A BYTES in the following places (the word A is used in the F83 model to select the alternate screen): screen 35 line 8, screen 36 line 2, screen 37 line 6, and screen 42 line 7.
7. Change all of the uses of the word ENDIF to THEN or use the definition of ENDIF given in screen 44 line 2.

The following is a better implementation of the word BYTES that appears in screen 21. This word now saves the current base and always uses HEX notation. Note that this word now uses the Forth-83 standard definition of WORD, so older systems will have to add the word HERE following WORD just as in the original.

```
:BYTES
    BASE @ >R HEX
    0 DO BL WORD NUMBER DROP
        C, LOOP
    R> BASE ! ;
```

The following is one possible method of implementing the word M68OUT in screen 18. This version saves the output code in an unused portion of memory. This block of memory can then be saved on disk or transferred in some other manner to the 68000 target machine. If you happen to be using the compiler on a 68000 system, you could execute the code directly from the host system.

```
M68K DEFINITIONS

( Pointer to current location in output
    buffer area of memory )
VARIABLE $M68OUT

( Store byte in output buffer area of
    memory )
: M68OUT ( b -- )
  $M68OUT @ C! 1 $M68OUT +! ;
```

The following code should be modified according to your needs and should be loaded after the rest of the cross compiler code.

```
( Start of output buffer space )
FORTH DECIMAL
HERE 10000 +
M68K DEFINITIONS
CONSTANT ($M68OUT)

( Initialize output buffer pointer )
($M68OUT) $M68OUT FORTH !
```

That sums up the changes I have made to the compiler. I am using the 68000 version of F83, which has Michael Perry's assembler built in. The assembler works just fine in conjunction with my cross compiler so people who want an assembler built into the compiler should take a look at the September 1983 issue of *Dr. Dobb's Journal*.

Sincerely,
Raymond L. Buvel
Box 3071
Moscow, ID 83843

Debugging Tool

To the Editors of *Dr. Dobb's Journal*,
A few weeks ago, I stumbled onto a tool that has greatly improved my debugging time. I primarily use Turbo Pascal and Forth in software development, and when a program crashes, I spend much of my time figuring out where it crashed, and why. I have evolved the code in the accompanying source listing [see Listing, page 12] to solve these problems. The necessary code to make an error trapping route is included.

The basic data structure is a stack of strings called *place*. The stack pointer is named *place_ptr*, and the stack depth is called *place_depth*, which is a constant.

To initialize the stack, call the procedure *starting*. At every level of invocation, such as the beginning of a procedure, function, or loop, call *entering* with the name of the place you are tracking. Upon exiting the procedure, function, or loop, invoke *leaving*. At the end of the section of the program you are tracking, call *ending*.

Let's consider the error handler. This procedure is passed a string of any acceptable length by its calling procedure, which is assumed to be an error message. After displaying the error message, which may be optionally printed on a printer, the place stack can be dumped either to the screen, the printer, or both. This indicates to any desired level of precision where in the program things went wrong and the invocation path as to how the program got there. It is then possible to type control-C and abort the program.

There are two innate errors that these procedures can trigger: stack underflow and stack overflow. The first problem is caused by invoking *leaving* more places than *entering*. This problem is detected by the procedure *leaving*, and the place stack as is will provide no useful information, because the place stack has been depleted prematurely. I find that the most common cause for this is placing *entering before* the loop starts and *leaving in* the loop. For example,

```
begin
   entering('loop_test');
   for i:=1 to 10 do
```

```
    begin
        do_some_stuff;
        leaving('loop_test');
    end;
end;
```

is an example of this kind of mistake. The *place* array still has the name *loop_test* in its first cell. By setting the *place_ptr* to *place_depth*, the dumping of the place stack will provide vital clues as to where the underflow occurred.

The second problem, stack overflow, is essentially the reverse situation: there were more invocations of *entering* than *leaving*. This problem can be detected two different ways, either while attempting to use *entering* or when *ending* has been invoked. In either case, the place stack dump will provide useful insights into what has been going on. When *ending* is encountered, the place stack should be empty; it if isn't, then there have been more invocations of *entering* than *leaving*, which indicates some fundamental failure of structuring in the program or in the placement of the *entering* and *leaving* invocations.

Earle Jennings
Director of R&D
Incremental Computing Corp.
P.O. Box 2875
Santa Clara, CA 95055

DDJ

(Listing begins on next page)

```
{**************************************************
**
**        Diagnostic tools in TURBO PASCAL
**          by Earle Jennings 10/21/84
**
*************************************************}
CONST
  name_length=24;  { length of names stored on the place stack}
  place_depth=50;  { the depth of the place stack }
TYPE
  name=string[name_length];
    { name is a string of name_length chars max}
  anystring=string[255];
VAR
  place_ptr:integer;
  place:array[0 .. place_depth] of name;
    { This is the place stack that provides our travel log}
{***************************************************
**  right_pad( Its_name,name_length )
***************************************************}
function right_pad(Its_name:anystring;name_length:integer ):anystring;
VAR i:integer;
  temp:anystring;
begin
  temp:=Its_name;
  if length(Its_name)<name_length then
  for i:=length(Its_name)+1 to name_length do temp:=concat(temp,' ');
  right_pad:=temp;
end; { of right_pad}
{**************************************************
**  dump_place_stack(target:text);          10/20/84
*************************************************}
procedure dump_place_stack(VAR target:text);
VAR i:integer;
begin
  writeln(target,'our present location is:');
  if place_ptr>-1 then
    for i:=place_ptr downto 0 do
    begin
      write(target,right_pad(place[i],name_length),',');
      if ( ( place_ptr-i) mod 3 ) = 0 then writeln(target);
    end;
end;  { of dump_place_stack }
{**************************************************
**  dump_places                            10/20/84
*************************************************}
procedure dump_places;
VAR it:integer;
begin
  writeln('dump place stack to: 1-crt, 2-lp, 3-both, else skip');
  readln(it);
  case it of
    1: dump_place_stack(con);
    2: dump_place_stack(lst);
    3: begin
         dump_place_stack(con);
         dump_place_stack(lst);
       end;
  else
  end;
```

(Continued on page 14)

Software Development Tools

If:

Manufacturer-Compatible, Fully-Supported Cost-Effective, Cross and Native Development tools are important to you!

Then:

Let us tell you about MICROTEC RESEARCH's extensive line of field-proven software development tools for serious software developers.

C
Pascal
PLM
Assemblers
Simulators
VT-100 Emulator
CPM/80 Emulator

Manufacturer Compatible

MICROTEC RESEARCH has been providing flexible and economical solutions for software developers since 1974. We've grown with the industry as our cross software development tools have transformed many large and small computers into powerful cross development systems for microprocessor applications. Our software tools are manufacturer compatible. Therefore, software made using manufacturers development systems may be more productively used in the MICROTEC RESEARCH cross-development environment.

Fully-Supported

MICROTEC RESEARCH's products are continually maintained and updated based upon the experience of thousands of installations worldwide. Revisions and modifications are distributed to licensees in warranty or covered by a service contract. Upgrades, which are major enhancements to a product's capabilities, are available at a significant discount. Our update/upgrade policy assures users they'll always be current with the fast moving world of microprocessor development.

Effective Differences

Beginning with product concept, through development, quality assurance, and post-sales support — **Quality, Compatibility,** and **Service** are the differences which set MICROTEC RESEARCH apart from the others.

Start Saving Time & Money

If you are a serious software developer, shopping for software development tools, call or write today for more information. **800-551-5554** In CA call **(408) 733-2919**

Target Microprocessors		Host Computers	Software Tools
8086/80186	68000/08/10	DEC VAX, PDP-11	C Compilers
8096	6809	DG MV-Series	Pascal Compilers
8080/8085	6800/01/02	DG Eclipse	PLM Compiler
8051	6805	Apollo	Assemblers
8048	6301	UNIX Systems	Linking Loaders
Z8002	6305	IBM PC	Librarians
Z80	6500	Data General/ONE	Download Communications
Z8	G65SCXX, G65SC1XX	HP 150	VT-100 Emulator
NSC 800	R65C00, R6500/1...	DEC Rainbow	CPM/80 Emulator
9900...	others	others	

for Serious Software Developers

MICROTEC™
RESEARCH

3930 Freedom Circle, Suite 101, Santa Clara, CA 95054
Mailing Address: P.O. Box 60337, Sunnyvale, CA 94088
(408) 733-2919 • Telex (ITT) 4990808

100% FLAWLESS COPIES . . .

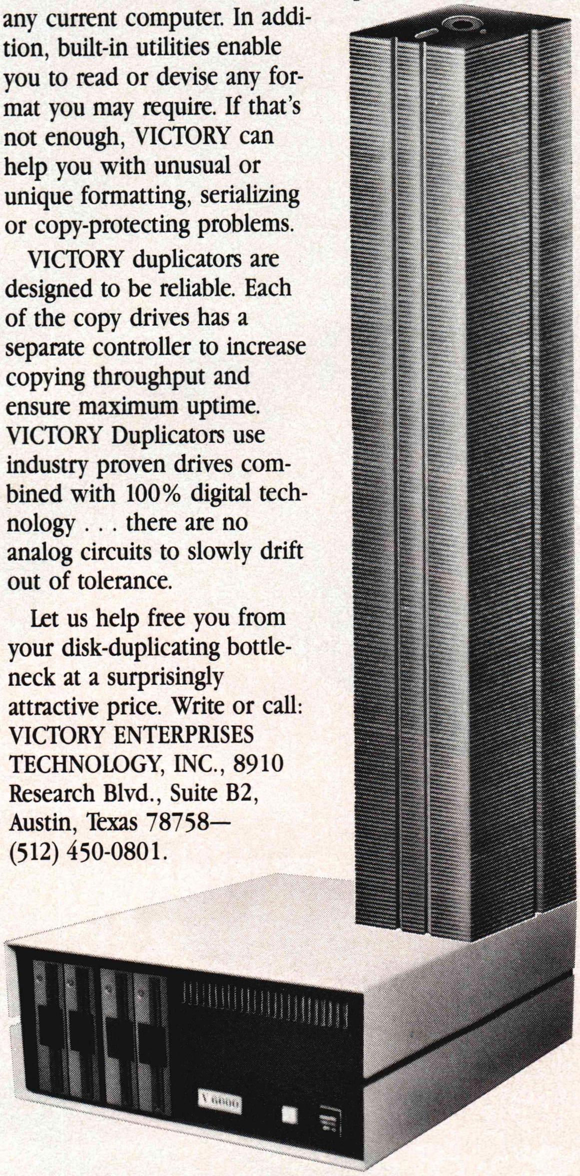
. . . FAST!

No need to tie up your valuable computer to duplicate diskettes . . . when VICTORY can provide you with a duplicator that will do the job flawlessly, and much faster. One button operation automatically formats, duplicates and verifies up to 8 diskette copies at the same time.

VICTORY can supply you with literally dozens of standardized formats to match the protocol of virtually any current computer. In addition, built-in utilities enable you to read or devise any format you may require. If that's not enough, VICTORY can help you with unusual or unique formatting, serializing or copy-protecting problems.

VICTORY duplicators are designed to be reliable. Each of the copy drives has a separate controller to increase copying throughput and ensure maximum uptime. VICTORY Duplicators use industry proven drives combined with 100% digital technology . . . there are no analog circuits to slowly drift out of tolerance.

Let us help free you from your disk-duplicating bottleneck at a surprisingly attractive price. Write or call: VICTORY ENTERPRISES TECHNOLOGY, INC., 8910 Research Blvd., Suite B2, Austin, Texas 78758—(512) 450-0801.



Victory Enterprises Technology, Inc.

Circle **no. 78** on reader service card.

SMALL C FOR IBM-PC

Small-C Compiler Version 2.1 for PC-DOS/MS-DOS
Source Code included for Compiler & Library
New 8086 optimizations
Rich I/O & Standard Library

$40

CBUG SOURCE LEVEL DEBUGGER FOR SMALL C

Break, Trace, and Change variables all on the source level
Source code included

$40

Datalight
11557 8th Ave. N.E.
Seattle, Washington 98125
(206) 367-1803

ASM or MASM is required with compiler.
Include disk size (160k/320k), and DOS version with order.
VISA & MasterCard accepted. Include card no. & expiration date.
Washington state residents include 7.9% sales tax.
IBM-PC & PC-DOS are trademarks of International Business Machines
MS-DOS is a trademark of Microsoft Corporation.

Circle **no. 31** on reader service card.

Only $95 with FULL SOURCE CODE!

Q/C

"... an incredible learning tool." *Byte*

For only $95, Q/C is a ready-to-use C compiler for CP/M with complete source code. Here's what *BYTE* (May 1984) said: "Q/C ... has a portable library and produces good code quality. If you want to learn compiler construction techniques or modify the standard language, Q/C is the obvious choice."

- Source code for compiler and over 75 library functions.
- Strong support for assembly language and ROMs.
- No license fees for object code.
- Z80 version takes advantage of Z80 instructions.
- Q/C is standard. Good portability to UNIX.

Q/C has casts, typedef, sizeof, structure initialization, and function typing. It is compatible with UNIX Version 7 C, but doesn't support long integers, float, parameterized #defines, or bit fields. Call about our new products: Q/C profiler, Z80 code optimizer, and Z80 assembler and virtual linker, all with full source code!

THE CODE WORKS
5266 Hollister, Suite 224
Santa Barbara, CA 93111
(805) 683-1585

Q/C, CP/M, Z80, and UNIX are trademarks of Quality Computer Systems, Digital Research, Zilog, Inc., and Bell Laboratories respectively.

Circle **no. 106** on reader service card.

```
end;  { of dump_places }
{**************************************************
**  error(message)                         11/3/84
**************************************************}
procedure error(message:anystring);
VAR y_or_n:char;
begin
  writeln('*****************************************');
  writeln(message);
  writeln('Do you wish error message output to printer?');
  readln(y_or_n);
  y_or_n:=upcase(y_or_n);
  if y_or_n = 'Y' then writeln(lst,message);
  dump_places;
  writeln('--->>>>  hit Ctrl-C to kill program, else return to continue');
  readln(y_or_n);
end;  { of error }
{**************************************************
**  enterring(it)                          11/3/84
**************************************************}
procedure enterring(it:name);
begin
  if place_ptr<place_depth then
  begin
    place_ptr:=place_ptr+1;
    place[place_ptr]:=it;
  end
  else error('place stack overflow upon enterring "'+it+'"');
end;  { of enterring }
{**************************************************
**  leaving                                11/3/84
**************************************************}
procedure leaving;
begin
  if place_ptr<0 then
  begin
    place_ptr:=place_depth;
    error('place stack underflow');
  end
  else place_ptr:=place_ptr-1;
end;  { of leaving}
{**************************************************
**  starting                              10/20/84
**************************************************}
procedure starting;
begin
  place_ptr:=-1;
  enterring('main');
end;  { of starting }
{**************************************************
**  ending                                 11/3/84
**************************************************}
procedure ending;

begin
  leaving;
  if place_ptr<>-1 then
  begin
    error('place stack imbalance');
  end;
end;  { of ending }
```

End Listing

D.E. Cortesi

A Random Choice

The generators of pseudo-random numbers are algorithms that lie on the boundary between computer science and recreational mathematics. They're as little tainted by real-world considerations as we could want, but they are endlessly measurable, improvable, discussable.

These thoughts were inspired by a recent issue of *PC World*, in which one Gary Andrew reported an unusually simple algorithm for generating random integers. Andrew gives credit to Drs. Wilson Talley and Nicholas Metropolis for the design of the algorithm. He claims it stands up well to a variety of tests, a claim we've not attempted to verify.

Andrew gives the algorithm in BASIC; we recoded it in C by way of understanding it. Our interpretation appears in the Listing (page 20). It looks to us as if a version that produced 16-bit numbers would be most effectively coded in assembly language. Then some tricks could be played with carry flags, etc., to avoid having to carry long-integer intermediate results. Anyone who codes one up, or who does any statistical tests of this generator, be sure to write.

Skies Still Falling, Ho-Hum

A couple of readers had comments on our November review of *Computers In Crisis.* Charles Martin of Durham, NC, comments: "The same issue—oh-migod what will happen to our Julian-date routines on 1 Jan 2000—was covered rather completely (read: thrashed half to death) in *ComputerWorld* in about 1975. That someone would write a book about it now is surprising; that Petrocelli would publish it only proves you don't have to be smart to be an editor . . . ."

Terry Jackson of Lombard, IL, with tongue firmly implanted in cheek, wrote: "The issue of date storage should be addressed promptly." Why? Because "it is well known to anyone associated with large DP operations that 16 years is not enough for them to solve this problem." But he has an answer: "The obvious solution is to get the whole world to agree to write the year in hexadecimal. That defers the crisis until 19FF (2155 old style), thus allowing time to rewrite the software, and *maybe* enough time to debug."

Throughputting

In the July Clinic we wrote about the idea of "throughput" and suggested that the throughput of a system's fundamental file-copy utility would be a useful capsule measurement of the performance of that configuration of software and hardware. We measured the time it took CP/M Plus and PIP to copy files of different sizes on three different hardware configurations. The numbers seemed to fit a linear model pretty well.

A dozen readers took the time to measure their own systems' copy times and report them. Their names, and the

Case-name	Machine	CPU	DOS	Drives and other factors
Cage	NorthStar	Z80 4MHZ	CP/M 2.2a	5-inch
Castle-1	NorthStar	Z80 4MHZ	N*DOS	5-inch
Castle-2	NorthStar	Z80 4MHZ	CP/M 2.2a	5-inch
Chamberlin-1	MTU-140	6502 1MHZ	CODOS	8-inch, 5K buffer
Chamberlin-2	MTU-140	6502 1MHZ	CODOS	8-inch, 32K buffer
Cortesi-1	S-100	Z80 4MHZ	CP/M 3.0	8-inch thinline
Cortesi-2	TRS-80 4P	Z80 4MHZ	CP/M 3.0	5-inch thinline
Cortesi-3	Apple-ALS	Z80 6MHZ	CP/M 3.0	Apple 5-inch
Floyd-1	PC	8088	PCDOS 2.0	5-inch, verify off
Floyd-2	PC	8088	PCDOS 2.0	5-inch, verify on
Floyd-3	PC	8088	PCDOS 2.0	5-inch to Mdisk
Floyd-4	PC-XT	8088	PCDOS 2.0	5-inch, verify off
Floyd-5	PC-XT	8088	PCDOS 2.0	5-inch, verify on
Floyd-6	PC-XT	8088	PCDOS 2.0	5-inch to Mdisk
Gunn-1	NorthStar	Z80 4mhz	CP/M 2.2	5-inch Tandon 100-2
Gunn-2	NorthStar	Z80 4mhz	CP/M 3.0	5-inch Tandon 100-2
Gunn-3	Zilog MCZ-1	Z80 2.5mhz	CP/M 2.2	8-inch Shugart 800-2
Hoffman-1	S-100	Z80 4MHZ	CP/M 2.2	8-inch single-density
Hoffman-2	S-100	Z80 4MHZ	CP/M 2.2	8-inch double-density
Hole-1	CompuPro	8088 8MHZ	MP/M 8-16	8-inch
Hole-2	CompuPro	8088 8MHZ	MP/M 8-16	21MB hard disk
Hole-3	CompuPro	8088 8MHZ	MP/M 8-16	Mdrive-H
Johnson	Morrow MD-3	Z80 4MHZ	CP/M 2.2	5-inch
Prince	n.a.	Z80 2MHZ	CP/M 2.2	8-inch single density
Sabin-1	Altos 5	Z80 4MHZ	CP/M 2.2	n.a.
Sabin-2	TRS-80 III	Z80 2MHZ	CP/M 2.2	5-inch
Schemm-1	Digital Grp	n.a.	CP/M 2.2?	8-inch
Schemm-2	Digital Grp	n.a.	Disk-MON	8-inch
Schemm-3	IMS	n.a.	CP/M 2.2?	8-inch
Schemm-4	NorthStar	Z80 4MHZ	CP/M 2.2	5-inch
Schemm-Z1	Z-110	8086	ZDOS	5-inch
Schemm-Z2	Z-110	8086	CP/M-86	5-inch
Swearingen	Big Board	Z80 4MHZ	CP/M 2.2	8-inch

Table 1
The cases—reporters and configurations—analyzed in the throughput study.

essential details of their hardware and software configurations, are listed in Table 1 (page 16). Most cases describe CP/M systems, but we received a few reports on NorthStar DOS, CODOS (what?), and PCDOS. No measurements of 68K-based systems, Apple DOSses, Triss-DOSses, Commodore 64s, or Ataris were reported. Where is everybody?

Table 2 (below right) lists the copy times reported with each case in Table 1. As shown, the numbers appear to be precise to the nearest tenth of a second, but this was *not* the case. Most times were taken with hand-operated watches, and so are accurate to the nearest 0.2 second at best (it is not difficult to get repeatable times to this accuracy using a hand-operated watch, but it is difficult to be any better). Some were reported only to the nearest sejod and were entered with a fraction digit of zero. And there may be a few transcription errors, since some numbers were submitted in minuscule handwriting on reader service cards!

Furthermore, we can't be sure that the measurements were taken in a consistent way. They were supposed to reflect the time to copy a file from one disk to another, exclusive of any time needed to load a copy program (the built-in COPY command of MSDOS; multiple-copy mode of CP/M PIP). There was an unstated (but obvious) assumption that all buffered data should be forced to disk in the measured time. We suspect that some sort of buffering phenomenon is affecting the more remarkable ZDOS and PCDOS measurements.

Throughput Models

Several readers made thoughtful comments on the numbers. Bruce Johnson of Lakewood, CO, had this to say: "The throughput you showed in the table is really a combination of two factors. One is the speed at which PIP transfers data—which I would call throughput—and the other is a theoretical minimum time to transfer a zero-length file, which I would call the setup time. The setup time corresponds to the Y-intercept of each line on the graph, while the throughput corresponds to the reciprocal of the slope of the line. I would suggest as a measure of throughput for linear utility programs:

$$T = 60/(T64 - T4)$$

where T = throughput, T64 = seconds to transfer a 64K file, and T4 = seconds to transfer a 4K file. This formula will yield a useful measure of throughput over the range of file sizes of most interest with only two measurements."

Johnson's ingenious formula does just what he claims; it produces numbers that are in fair agreement with a more elaborate model to be discussed below.

William Hole, of Barrington, RI, made the same point, that the timings we published and those he took on his own system were a good fit to a linear model with a fixed startup time. He put his numbers through a linear regression test to check the model and got good agreement.

CP/M Measurements

Because Bill Hole seemed to know his way around the fundamental tools of statistics, we dumped the entire contents of Table 2 on him and asked him to test them for conformity to the following model:

$$T = s + e * \text{floor}(n/16) + n/t$$

where T is the reported time in seconds, s is a setup time, e is the time required by CP/M for opening the subsequent "extent" of a file, n is the number of kilobytes that are transferred, and t is the number of kilobytes that can be transferred in one second, that is, the actual "throughput" of the system.

He performed this analysis for us (and many thanks for the time it must have taken) but the results are somewhat equivocal. Table 3a (page 18) reports the analysis for the CP/M-compatible systems. The first group of three columns presents the computed results for the setup time. The estimat-

	1k	2k	4k	8k	16k	24k	32k	48k	64k	128k
Cage	3.8	4.8	6.6	10.5	18.8	-.-	36.2	53.2	72.0	151.1
Castle-1	5.5	5.8	7.9	11.5	20.0	-.-	40.0	-.-	78.0	
Castle-2	6.0	6.4	7.2	9.0	16.6	-.-	29.4	-.-	56.7	
Chamberlin-1	2.4	2.2	3.1	3.8	4.7	6.4	7.6	10.6	13.6	23.4
Chamberlin-2	2.5	2.5	2.9	3.6	4.6	5.6	6.5	6.5	10.4	18.5
Cortesi-1	3.4	3.4	3.4	4.8	5.8	8.2	9.6	13.2	18.0	
Cortesi-2	3.2	4.0	5.0	7.2	11.0	17.4	20.6	30.0	39.5	
Cortesi-3	4.0	4.2	5.0	6.4	9.2	13.8	17.4	23.6	32.5	
Floyd-1	2.3	2.3	2.8	3.4	4.1	5.1	5.8	7.3	12.5	
Floyd-2	3.5	3.5	4.2	4.8	5.9	7.4	8.4	10.6	17.5	
Floyd-3	1.3	1.3	1.7	2.0	2.4	3.1	3.3	4.1	6.3	
Floyd-4	4.3	4.6	4.7	5.2	5.3	5.3	5.4	6.2	7.5	
Floyd-5	4.8	4.9	5.0	5.6	5.7	5.9	6.4	7.3	8.7	
Floyd-6	3.1	3.5	3.5	3.6	3.9	3.9	4.0	4.5	5.4	
Gunn-1	2.6	2.6	2.9	3.8	5.6	7.6	9.2	14.2	18.6	
Gunn-2	3.5	3.6	4.1	4.7	6.7	8.0	9.7	14.1	17.9	
Gunn-3	2.2	2.4	3.0	4.2	6.0	8.9	11.2	17.3	22.6	
Hoffman-1	4.0	5.0	6.0	9.0	16.0	25.0	31.0	42.0	55.0	
Hoffman-2	-.-	2.5	3.5	4.0	6.0	10.0	13.0	16.5	22.0	
Hole-1	2.9	3.1	3.6	3.9	5.5	7.7	9.8	13.5	17.0	33.5
Hole-2	2.1	2.2	2.4	3.5	6.4	6.9	8.7	13.3	16.4	30.8
Hole-3	0.6	0.6	0.7	0.9	1.3	1.7	2.1	2.9	3.8	7.0
Johnson	3.6	3.7	4.7	6.0	9.6	12.2	17.1	23.5	31.7	
Prince	-.-	2.4	2.6	3.4	5.6	-.-	11.5	-.-	22.7	
Sabin-1	2.5	3.5	5.0	7.0	12.5	17.0	23.0	48.0	44.0	
Sabin-2	2.5	3.0	4.5	6.0	9.0	11.5	17.0	22.5	31.0	
Schemm-1	-.-	4.9	5.2	5.9	8.5	-.-	13.8	-.-	24.2	
Schemm-2	-.-	2.6	2.6	3.4	3.7	-.-	5.8	-.-	-.-	
Schemm-3	-.-	7.6	8.0	9.7	13.8	-.-	23.3	-.-	42.8	
Schemm-4	-.-	6.2	7.2	7.3	9.4	-.-	14.8	-.-	22.9	
Schemm-Z1	-.-	1.1	0.9	1.1	1.3	-.-	1.6	-.-	2.9	
Schemm-Z2	-.-	3.1	3.1	3.2	4.5	-.-	6.8	-.-	12.0	
Swearingen	3.4	3.6	4.2	5.8	8.2	-.-	15.9	20.9	28.3	

Table 2

Reported times, in seconds, to transfer files of different sizes. These data were input to the analysis performed by W. T. Hole. The two italicized entries were mistranscriptions (see text).

ed setup time appears in the first column. The second column (headed "s/e") gives the standard error in seconds; that is, the estimated setup time, plus or minus this value, should encompass 68% of the observations.

The third column, headed "prob," is a confidence rating. It is the probability that the setup value should equal zero. If it exceeds 0.05, the setup time is probably not a reasonable part of the model. The eye is drawn at once to the third line, case Sabin-1, the only case that appears not to conform to a model that includes a fixed setup time. Why is this? Quite possibly because, as indicated in Table 2 by an italicized entry, one of Joe Sabin's numbers was mistranscribed (the 48K copy time should be 34.0 seconds). There wasn't time to rerun the analysis, so we left this case in place as an interesting exception. The fact that the only case not to fit the model contained bad data seems to strengthen the case for the model.

The second group of three columns in Table 3a represents the estimate for factor "e," the time to open a new 16K file extent. Again the first column is an estimate of the time, the second is the standard error—read it as "plus or minus seconds"—and the third a confidence rating. You'll immediately notice that the second line exhibits a remarkably unconfident probability of 0.967. A 16K extent overhead isn't apparent in the numbers for this case (which were transcribed correctly). That's odd, because (as we can see in Table 1) this case involved CP/M single-density 8-inch disks. Other cases that disagree with the hypothesized extent factor are easier to explain, e.g., because they use a 32K extent or (in the case of the M-drive) because the extent overhead is actually near zero.

The final group of three columns presents the estimated throughput, with the same interpretation as before. A poor confidence rating here (one greater than 0.05) indicates that these data do not fit a linear model. There is one case in Table 3a for which this is definitely true and two others that are suspect.

But by and large, the linear model of throughput fits very well, so the throughput value (on which Table 3a is sorted) is probably a fairly good estimate for these CP/M-based systems. It shows a general trend of increasing speed from 5-inch drives to 8-inch to M-drives, with Hole's CompuPro M-drive the fastest. Schemm's Z-100 (8-bit CP/M, but with I/O handled by a 16-bit BIOS) produced a remarkable time for a 5-inch disk drive. We suspect this measurement was affected by some sort of internal buffering—notice that it conforms less well to the linear model than most.

Non-CP/M Throughput

Table 3b (page 18) contains the analy-

Case	--- Startup ----			--- Extent -----			--- Throughput -----		
	sec.	s/e	prob	sec.	s/e	prob	kb/sec	s/e	prob
Cage	3.13	0.92	0.014	5.17	1.58	0.017	1.15	0.14	0.0001
Hoffman-1	3.32	0.75	0.004	-.07	1.73	0.967	1.21	0.14	0.0001
Sabin-1	1.26	3.23	0.710	-.72	7.40	0.924	1.27	1.10	0.0747
Castle-2	4.67	0.55	0.001	2.36	1.23	0.126	1.42	0.14	0.0003
Cortesi-2	3.06	0.24	0.000	1.50	0.54	0.033	1.99	0.11	0.0001
Schemm-3	6.36	0.24	0.000	2.47	0.48	0.014	2.20	0.13	0.0004
Sabin-2	2.08	0.49	0.005	-.22	1.13	0.848	2.21	0.31	0.0002
Johnson	2.76	0.38	0.000	0.43	0.88	0.638	2.36	0.27	0.0001
Cortesi-3	3.52	0.29	0.000	1.86	0.67	0.032	2.79	0.28	0.0001
Swearingen	2.93	0.34	0.000	0.93	0.77	0.280	2.86	0.35	0.0003
Hoffman-2	2.17	0.55	0.011	0.82	1.12	0.493	3.64	0.96	0.0048
Hole-2	1.52	0.26	0.000	1.72	0.47	0.165	3.73	0.41	0.0001
Schemm-1	4.14	0.22	0.000	1.00	0.44	0.106	3.74	0.35	0.0013
Gunn-3	1.88	0.19	0.000	1.41	0.44	0.019	3.90	0.37	0.0001
Prince	1.70	0.18	0.002	1.82	0.36	0.015	4.10	0.34	0.0010
Schemm-4	5.81	0.46	0.001	0.59	0.91	0.559	4.12	0.99	0.0139
Gunn-2	3.14	0.22	0.000	0.52	0.50	0.339	4.93	0.69	0.0002
Gunn-1	2.16	0.21	0.000	1.22	0.49	0.047	5.10	0.73	0.0002
Hole-1	2.68	0.17	0.000	1.19	0.31	0.006	5.75	0.63	0.0001
Cortesi-1	3.08	0.21	0.000	1.35	0.49	0.033	6.11	1.07	0.0005
Schemm-Z2	2.70	0.20	0.000	0.94	0.39	0.096	9.96	2.50	0.0157
Hole-3	0.51	0.01	0.000	0.04	0.03	0.258	20.61	0.80	0.0001

Table 3a

Analysis of the CP/M (and MP/M) cases, ranked by increasing throughput.

Case	--- Startup ----			--- Extent -----			--- Throughput -----		
	sec.	s/e	prob	sec.	s/e	prob	kb/sec	s/e	prob
Castle-1	3.96	0.32	0.000	3.24	0.72	0.010	0.99	0.03	0.0001
Chamberlin-1	2.17	0.21	0.000	-.28	0.37	0.473	5.45	0.71	0.0001
Chamberlin-2	2.50	0.44	0.000	0.42	0.78	0.607	10.03	7.92	0.0580
Schemm-2	2.43	0.21	0.007	0.64	0.62	0.409	11.80	4.38	0.0660

Table 3b

Analysis of 8-bit DOS's other than CP/M. These cases were a poor fit to the hypothesized model.

Case	--- Startup ----			--- Extent -----			--- Throughput -----		
	sec.	s/e	prob	sec.	s/e	prob	kb/sec	s/e	prob
Floyd-2	3.44	0.67	0.002	1.80	1.54	0.286	9.00	19.73	0.1959
Floyd-1	2.27	0.54	0.005	1.26	1.23	0.345	12.26	36.28	0.2291
Floyd-3	1.35	0.20	0.000	0.34	0.48	0.500	18.24	13.81	0.0594
Floyd-5	4.79	0.15	0.000	0.06	0.35	0.853	18.72	9.05	0.0222
Floyd-4	4.47	0.20	0.000	0.01	0.47	0.974	24.50	32.31	0.1303
Floyd-6	3.28	0.13	0.000	0.01	0.30	0.956	35.08	38.44	0.1043
Schemm-Z1	0.97	0.12	0.004	0.33	0.23	0.252	72.99	515.24	0.3383

Table 3c

Analysis of the few reported PC-, MS-, or Z-DOS cases. Not only the extent, but the linear model is rejected by these numbers.

sis of the cases that involved 8-bit systems other than CP/M. Mostly these deny an "extent" hypothesis—not unreasonably, since that's a CP/M peculiarity—and half of them appear to be nonlinear. One of these is due to a transcription error: we miscopied Hal Chamberlin's 48K time as 6.5 seconds when it should have been 8.3 seconds. With that change, case Chamberlin-2 should be much closer to linearity, but time didn't permit rerunning the analysis.

Table 3c (page 18) details the analysis for the MSDOS (PCDOS, ZDOS) cases. We expected that an MSDOS copy command would be just as linear as a CP/M PIP operation. That appears not to be so, as *none* of these cases conforms to the linear model. That they don't support the "extent" factor is no surprise; there are no "extents" in MSDOS. But that they don't appear linear is most puzzling, as is the remarkable speed they suggest. (After all, no amount of software cleverness will make a 5-inch drive rotate or seek any more quickly, so why should a PC be so *very* much quicker than an 8-bit system with equivalent drives?) Can a reader suggest what might be going on? Is there a measurement procedure for MSDOS that will produce linear numbers?

The Johnson Model

We present Table 4 (below) as a final view of the data. It shows all the cases that support a linear model, ranked by what we might call their "Johnson Number"—the figure of merit produced by Bruce Johnson's formula discussed earlier. There is a fairly close correspondence, so Johnson's rule of thumb is useful *when the data is linear.* But since it presumes linearity, it should only be applied where linearity is known to exist.

DDJ

(Listing begins on page 20)

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 190**.

Case	4k Time	64k Time	Johnson Number	Linear Model
Castle-1	7.9	78.0	0.86	0.99
Cage	6.6	72.0	0.92	1.15
Castle-2	7.2	56.7	1.21	1.42
Hoffman-1	6.0	55.0	1.22	1.21
Schemm-3	8.0	42.8	1.72	2.20
Cortesi-2	5.0	39.5	1.74	1.99
Cortesi-3	5.0	32.5	2.18	2.79
Johnson	4.7	31.7	2.22	2.36
Sabin-2	4.5	31.0	2.26	2.21
Swearingen	4.2	28.3	2.49	2.86
Prince	2.6	22.7	2.99	4.10
Gunn-3	3.0	22.6	3.06	3.90
Schemm-1	5.2	24.2	3.16	3.74
Hoffman-2	3.5	22.0	3.24	3.64
Gunn-1	2.9	18.6	3.82	5.10
Schemm-4	7.2	22.9	3.82	4.12
Cortesi-1	3.4	18.0	4.11	6.11
Hole-2	2.4	16.4	4.29	3.73
Gunn-2	4.1	17.9	4.35	4.93
Hole-1	3.6	17.0	4.48	5.75
Chamberlin-1	3.1	13.6	5.71	5.45
Schemm-Z2	3.1	12.0	6.74	9.96
Hole-3	0.7	3.8	19.35	20.61

Table 4
The cases that matched a linear model, ranked by Bruce Johnson's rule-of-thumb formula.

Dr. Dobb's Clinic Listing (Text begins on page 16)

```
/*
        Random integer algorithm reported by Gary Andrew to
        the "*.*" column of PC World, December 1984, with
        credit to Drs. Wilson Talley and Nicholas Metropolis.
        Recoded to C by DEC.

        This version produces 15-bit integers; for 16-bit
        unsigned integers longs must be used for intermediate
        computations.
*/
#define BITS 15 /* how many bits in numbers */
#define MAX0 1 << BITS
#define MAX (MAX0) - 1

#define private static
#define cardinal unsigned /* is this Wirth-while? */

private cardinal seedA = 32749, seedB = 32633;

int rseed(a,b)
cardinal a,b;
{ /* seed the generator, returning first number of sequence */

        a &= MAX;           /* reducing negative signed ints, */
        b &= MAX;           /* ..etc., to values <= MAX */

        a *= 2; if (a>MAX) a -= MAX;
        b *= 2; if (b>MAX) b -= MAX;

        seedA = a; seedB = b;
        return (b);
}

private cardinal rand0()
{ /* cycle the generator, return next number of sequence */

        register cardinal c;

        c = seedA + seedB;
        if (c>MAX) c -= MAX0;
        c *= 2; if (c>MAX) c -= MAX;
        seedA = seedB; seedB = c;
        return (c);
}

cardinal randi(n)
cardinal n;
{ /* return a random number in 1..n */

        register cardinal c;

        c = rand0();
        return ( 1+ (c % n) );
}
```

End Listing

C/UNIX PROGRAMMER'S NOTEBOOK

by A. Skjellum

Before beginning this month's column, I want to express my congratulations to *DDJ* on its 100th issue. (My first *DDJ* article appeared a little more than five years ago.) I hope that readers will continue to make their excellent contributions to keep the magazine going during the next few years; I include both members of the "old guard" and new readers in that wish.

This month, I include more of the reader responses received over the past few months—plus some ideas about variable formats in printf( ) functions.

A C Style Reference

Last year, we had a long-running discussion on C style. Several references were cited at that time. Since then, I've run into another paper on the subject. While the formatting is different from what I prefer, I think the paper is well worth reader attention. The article, called "A C Style Sheet," was written by Martin Minow of Digital Equipment Corp. The article lists an address, so perhaps this is the best way to acquire a copy:

Martin Minow
Digital Equipment Corp.
146 Main St., MLO 3-3/U8
Maynard, MA 01754

It includes a list of references from which the work has been abstracted.

Wish List

John M. Gamble of Batavia, Ohio, writes:

"I am very glad you printed this column [August 1984] (I ranked it number one for the month) because it inspired me to jot down my own wish list. It is only one wish long, but it is something I have longed for for some time."

Mr. Gamble's wish is for a way to control more precisely the sizes of intrinsic data types in C. For example, integers can have different sizes on different machines. He continues:

"The size of the various types (int, char, long, etc.) vary too much from machine to machine. However, I don't think that forcing a size to a type (a la Plum Hall) is the answer (besides, I hate the baby words Plum uses to define them). Instead, I think that another 'storage class specifier' (see *The C Programming Language*, p. 192) like typedef could be defined. I'm going to call it 'sizetype.' Sizetype would be used just like typedef, but what is declared is the size of the storage class. For example, we could define a type small this way:

```
sizetype u8 small;
```

The letter 'u' is optional and stands for 'unsigned.' Thus, a variable of type small is unsigned and 8 bits long. Another example would be:

```
sizetype 16 hexsize;
```

which would guarantee that variables of type hexsize are 16 bits long."

I am very enthusiastic about Mr. Gamble's suggestion. It would enhance portability of C code. Anyone who has moved between microcomputers and minis is aware that significant problems can occur when moving from machines with signed or unsigned characters. Integer length differences are also annoying. He adds:

"Use of the sizetype declaration would guarantee portability between machines (which currently is a heck of a problem). It also means that only one more reserved word is added, instead of the many which would be needed to define a type for every conceivable integer length. If the sizeof operator could be altered to return fractions, we could use sizetype to define bit lengths that are not multiples of eight. On the other hand, I notice that the sizeof operation on a bit-field structure is not defined in *The C Programming Language*, so maybe it is just up to the person who writes the compiler."

Another August Response

Readers may have followed the somewhat fervent comments made by Gerald Evenden in the August 1984 column. He has responded to my remarks, and these remarks deserve further commentary. To summarize the previous material: Mr. Evenden was extremely displeased by my comments about C because he felt that they could convey the wrong impressions to the uninitiated. Furthermore, he felt it unfair for me to discuss C I/O libraries without strongly emphasizing that C and its libraries are completely separate concepts (which they indeed are). He now writes:

"In regard to your response to my letter in the August issue . . . I feel I must expand upon some of my earlier points and make some additional comments. In addition, please excuse the excesses of a middle-age curmudgeon. Scars acquired in numerous battles of the computer wars tend to create a knee-jerk reaction when I sense some potentially deviant and dangerous thought processes. I wanted to emphasize that the compiler and the support library are two very distinct entities and we must be careful to maintain the distinction. When I talk of C, I am referring to the compiler . . . when I talk about the C library, I am referring to what I feel is currently a very vague and poorly defined item."

The original concept of a C library is defined in *The C Programming Language*. Other libraries are available on Unix systems, but these vary from installation to installation and from version to version. I agree with Mr. Even-

den that the words "C library" are currently vague. In my opinion, we should standardize functions that are not inherently Unix-only features. For example, we should include:
(1) "block" input-output (read, write, and relatives)
(2) stream input-output functions (e.g., fopen, fwrite, fputc)
(3) memory allocation functions (calloc, malloc, . . .)
(4) setjmp/longjmp procedures
(5) alarm (but not signal, which is Unix dependent)
(6) exit
(7) scanf, printf, and relatives
Furthermore, we should include the Unix math library, because these are fundamental routines (e.g., sin( ), exp( )). Readers may wish to formulate an exact list for exclusion and inclusion. If there is sufficient response, we could propose a standard for the C library in a future column.

Mr. Evenden continues:

"This sensitivity to the compiler-library problem is caused by having to deal with compilers where too many features that should have been relegated to the support library were included as part of the compiler [i.e., language definition]. For example, Fortran gives us READ, WRITE, and a few other input-output support operations, which must be treated by the compiler as special operations since the syntax does not match normal external module calls (of course, external modules are involved, but they are transparent to the programmer). When specialized input-output is required, which can't be handled by these statements, all sorts of contortions are done by the programmer to get around these restrictions . . . typically these gyrations are specific to the host system. In addition, many manufacturers will compound a bad situation by supplying a compiler with supplementary functions to provide access to unique features of their system. Goodbye transportability!"

Mr. Evenden is one hundred percent correct. Fortran's implicit connection of input-output functions to the language is a terrible failing. Pascal also suffers from this malady, even though it is a newer structured language. Evenden continues with the following remarks:

"In the case of C, the compiler writer doesn't have to go out of his way to handle special input-output syntax, and the programmer utilizing the typical C library can go to basically three levels of input-output to handle his problem:
(1) basic block 'read' – 'write'
(2) buffered (stream) input-output with the getc( ), putc( ) functions
(3) Fortran-like scanf( ) and printf( ) operations
This is an excellent example of building-block code: the read-write level is the lowest level and is the only place where we have to deal with the host machine's operating system; each successive level uses the previous level's entries. The applications programmer can thus choose the starting level best suited to his job and add the remaining tiers of code to perform his task."

Given this buildup of the C library, Mr. Evenden returns to the reasons for his original objections:

"One of the principal fears I have is that if we get to talking about the C compiler and a standard library in one breath, we will find some well-meaning ignoramus developing a C compiler with built-in input-output functions (or, for that matter, other 'special' features). In this situation, our input-output is engraved in stone, and we will be

forced back into the same situation involved in Fortran coding. With C, we can individually or collectively trash the input-output part of the library in favor of some new software and still preserve the compiler itself. In addition, the old software is still good as long as we maintain a working copy of the old library. We often cannot do this if the compiler has been rewritten. I would much rather try to transport a program where a few specialized routines had to be rewritten than . . . deal with compiler variations."

To summarize the principal points Mr. Evenden makes (and with which I agree):

(1) Computer languages like C are superior because they segregate their library from the language definition.

(2) Because of (1), the language offers greater maintainability, even through revisions of the libraries, because we can retain old libraries more readily than whole compiler environments.

(3) Specialized applications can completely ignore the standard software library without any loss of power.

(4) When informing/teaching people about C, we must emphasize this unique feature to ensure that it is retained in future incarnations of the language. We hope future languages will also be constructed in this way.

To summarize his point, Evenden states:

"The problem of C libraries and what is a 'standard' C function is not yet resolved and needs further discussion. Tight binding of C and Unix is unfortunate, and we need to dissociate the two if we are to encourage non-Unix use of C and transportable C software. An important part of this unbinding is specifying a viable C library which can be installed without ambiguity and omission on a wide range of operating systems."

This is similar to what I mentioned above. I encourage a reader effort along these lines.

Tongue Biting

In his previous letter, Mr. Evenden indicated that he usually "bit his tongue" instead of complaining about C. I suspect that he is truly concerned that dissent about C could lead to its demise as an important language. He addresses this point as follows:

"My 'tongue biting' remark related to criticizing the C language was, of course, mostly rhetoric. But most of my criticisms are minor except for one: evaluation of all floating-point data elements in double precision. When two type float values are joined by a binary operator, I fail to see any reason why we should pay the costly runtime premium of double-precision evaluation! Of course, conversion of float function arguments to double is equally strange and pointless. I suspect that floating-point arithmetic was one of the last features added to the language and got shortchanged in the final stages of development of C. If the people at Bell Labs have any excuse for this peculiar handling of floating point by C, I'd sure like to hear it! Hopefully, some . . . readers may also have comments."

Mr. Evenden has hit on an important point. Not only is the conversion of float to double expensive, programming can get messy when dealing with pointers and arrays. For example, a program that needs a large number of floats (in an array) is difficult to use with a function that expects pointers to double. I see no reason for using arrays of double-precision numbers for many applications. Yet, to simplify programming logic, memory must often be wasted (by a factor of two) for floating-point arrays. On systems like the 8086/8087, the actual cost for single/double-precision operations is the same, but conversions and other effects are still cause for grief.

Evenden continues:

"I have given some thought to what would be required to make C's floating point behave in a more traditional manner and have come to the conclusion that upward compatibility of a new compiler might not be possible as far as floating-point syntax is involved. Obviously, the 'standard' library routines will have to be changed ('printf' will need '%E' and '%e') . . . functions returning floating-point values will have to be in two precisions (sqrt( ) and dsqrt( ), for example)."

Variable Formats in printf( )

I'd like to change gears and consider an aspect of printf( ) format strings. Consistent with the previous discussion, I remind readers that this is a discussion relevant to the C library and not the compiler.

A typical printf( ) call might look as follows:

```
float number;
...
printf("%7.3e\n",number);
```

In certain cases, we might wish to vary the format dynamically. One way to accomplish this is shown in the Figure (page 25).

The format string in the sprintf( ) presented in the figure is "%%%sf." The first two percent signs place a single output percent in fmt_string. The "%s" causes the string format (containing "7.3") to go into fmt_string. Finally, the letter "f" is interpreted literally and is sent to fmt_string. The point of giving this example is to show how cumbersome the operation can be. Now I want to pose a simple solution to the problem.

To allow variable formats as part of a single printf( ), we need a way to indicate an indirection. Then printf( ) could use the current member of its argument list as a source for the format. This is illustrated as follows:

```
float number;
...
printf("%&s\n","7.3f",number);
```

The indirect format is "&s," which tells printf( ) to take the first argument as a string and print it into the format string before proceeding. Thus, the ultimate format string is "%7.3f." Another possibility would be:

```
float number;
float format = 7.3;
...
printf("%&f\n",format,number);
```

which demonstrates the range of choices allowed by indirect formats. To print an ampersand, we would need the following sequence:

```
printf("%&&");
```

Why would these be useful? Primarily, they allow programs to readily adapt to data variations. This could allow greater user selectability or, if extended to scanf( ), greater ability to

read and write "foreign" data files.

Conclusions and Comments

In this column, I have presented more reader feedback and some brief comments about printf( ).

The "C/Unix Programmer's Notebook" was started in September 1983. I think that I have achieved a lot in writing the column during this last year. Unfortunately, other responsibilities make it impossible for me to continue. Therefore, I wish to thank those readers who have responded for their assistance, ideas, and criticisms. I also hope the readership will provide the new columnist with the same level of enthusiastic support that I have received. Intentionally, I have left few loose ends in my discussion. This will allow the next columnist to develop areas of discussion without too much loss of continuity. With a new columnist comes a new perspective, and I hope that more Unix coverage will be possible. Certainly, I don't expect that C will be excluded for the benefit of Unix but rather that a balance will be struck. Perhaps someday we will have separate "Notebooks" for each subject.

Thank you. I look forward to enjoying the next one hundred issues of *DDJ*, and I hope you enjoy them too.

An Addendum: The Future of C

Concerning future upgrades/modifications of C, Evenden writes:

"Some people criticize C as being a 'Spartan' language, but I maintain that this Spartan attribute is its principal and strongest feature . . . [C] is a real programmer's language providing an excellent tool for doing everything from real-time processing, to writing other compilers, to sophisticated scientific applications. If we ever make changes to C, we will have to be very careful to maintain this strong feature of the language."

I leave this quotation as my parting remark.

DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 191**.

```
float number;
...
char fmt_string[100];                      /* make format string here*/
char format[10];                           /* format contained here */
strcpy(format,"7.3");                      /* copy a specific format */
...
sprintf(fmt_string,"%%%sf",format);        /* make format string */
printf(fmt_string,number);                 /* print number in format */
```

Figure

Festschrift for Doctor Dobb

Contributions by Suzanne Rodriguez, Tom Pittman and Bob Albrecht in celebration of the 100th issue of *DDJ*

Suzanne Rodriguez was editor of DDJ *from January 1979 to September 1980. It was Suzanne who introduced the funky colored border (designed by Betsy Roeth and Aleeca Harrison) and Suzanne who published Ron Cain's original Small-C compiler.*

Tom Pittman wrote to DDJ *in March 1976 to announce his version of Tiny Basic for the Motorola 6800 microprocessor. Unlike the* DDJ *version, Tom's Basic was not free ("software is my living," he said). The price: $5.*

Bob Albrecht is the "OB" of Dr. Dobb. After launching (with Dennis Allison) DDJ, *he went off to launch new projects. He is presently involved in a corporation dedicated to computers and small children, fantasy role-playing games, and tennis. His contribution here launches a new* DDJ *department by Bob and Mike Swaine: Realizable Fantasies.*

Who is Dr. Dobb? Can you hack on a Mac? And who are the hackers of tomorrow?

The Bob and Dennis Show
by Suzanne Rodriguez

In the past year I've read more than one book or article discussing the founding of *Dr. Dobb's Journal*, and chances are that you have, too. It all sounds fairly straightforward on the surface: Bob Albrecht asked Dennis Allison to write a 2K BASIC that could be published in the newspaper Albrecht had founded, *People's Computer Company*. Allison complied. The rest is history.

What's not history—at least not yet—is how incredibly prolific and creative Albrecht and Allison are. Together or individually, they've founded many successful organizations, and even a few publications, got them going and then gone on to pursue new ventures. Allison's ventures tend to take place in the business and academic worlds; Albrecht's almost exclusively with children and computer learning.

What is also not history is the long list of people whose lives have been influenced, at a formative time, by one or the other—or both—of this duo. If you've been following the routine rules and regulations of the world, doing things the way you've heard they're supposed to be done, your first interaction with Allison and/or Albrecht can leave you somewhat dumbfounded. Prolonged interaction either gets you running with a vengeance back toward the world of rules, or it changes you.

Take me, for instance. I first met Allison and Albrecht in late August of 1978, when they interviewed me to become the third editor of *Dr. Dobb's Journal*. (Jim Warren, *DDJ*'s first editor, had gone on to found both the *Computer Faire* and the newspaper that would, in time, become *InfoWorld*. Warren's successor, Tom Williams, was soon leaving *DDJ* to pursue other endeavors.) At the time I was young and inexperienced enough to be impressed with my own credentials, which were really rather slender: I had published a number of magazine articles; I had been a technical writer at Zilog, writing their first piece of documentation; and I was working on my Master's in Journalism at Stanford. The idea of being editor of *DDJ* was very exciting: I could combine what I'd

be learning in school with the technical aspects of the magazine.

At least that's what I thought. I imagined that I'd be hired for a straightforward, rational, logical job, and that I'd be working closely with rational and logical people, applying well thought-out publishing principles in a logical and rational manner. Boy, did I have a thing or two to learn!

First there was the interview with Allison. He greeted me in his antique-and-book-laden home and proceeded to question me very thoroughly about my recent trip to Greece. We sipped retsina and talked for an hour about Ios and Samos, ouzo and feta, sleeping on beaches and hitching rides on donkeys. Finally he turned to me and said: "Well, Suzie, as far as I'm concerned, you've got the job."

I was very puzzled. We hadn't mentioned the word *computer*, and the topic of *Dr. Dobb's Journal* had not come up at all. "But," I stammered, "don't you want to hear about my background? I mean, I've worked with . . ."

He dismissed my words with an impatient wave of his hand, a gesture which I would come to learn was characteristic. "No, no! I think *DDJ* will be in good hands. I make these decisions based on character, and it doesn't seem that too much will get by you. You might find it difficult working with Albrecht, though—he can be trying—but I think you can handle him."

I went both dazedly and apprehensively off to my interview with Albrecht. He had asked me to join him in a "piano bar safari." I had no idea what that was, but I was soon to find out. A *true* piano bar safari consists of going from one piano bar to another, drinking beer and singing in each. At our initial meeting, however, we only hit one piano bar. I managed to find him in the dark interior, and we spent the next few hours (when he wasn't singing) talking about fantasy role playing, Greek dancing and tennis. Once more, computers and *Dr. Dobb's* were never discussed. When it was time for me to go, Albrecht said, almost as if in afterthought: "Well, if Dennis says you're ok, that's good enough for me."

This time I didn't even bother to start flaunting my qualifications. "Fine," I said, "I'm looking forward to working with such a talented bunch."

"Oh, you'll learn a lot, there's no doubt about that," said Albrecht. "You might have trouble with Allison, though—he can be difficult—but I think you can handle him!"

This was all the beginning of something great—my two years at *DDJ*. Every day I would sit in my classes and learn about *the way it's supposed to be* in the publishing world. And then I would dash to *People's Computer Company* and experience *the way it really is.* Or at least the way it was in that particular place and at that particular time.

While the day-to-day decision making for *DDJ* was in my hands, larger decisions—for instance, should we go to glossy paper—were decided by the Board. This occasioned marvelous and stormy battles, the best of which were with Bob Albrecht. In the end he would always give in and tell me he'd wanted it all along. "I just like a good fight," he would say. This would usually be a day in which I had taken a mid-term in something like "Human Resources and Intra-Company Dynamics". If I'd used my real-life experiences with Albrecht as illustrative examples, I would have flunked the course.

Dennis Allison, on the other hand, never fought; he just told you exactly what he thought. "I think that's a fantastic idea" made you feel just great. "I

Circle **no. 43** on reader service card.

think that's the dumbest idea I ever heard" plunged you into the depths. I kept telling Dennis that what I had learned in my seminar on how to motivate others taught me that he was going about things all wrong. In response, he said that was the dumbest thing he'd ever heard.

There were the even stormier meetings of the Board of Trustees where Allison would bait Albrecht, or Albrecht would bait Allison, and one or the other would storm out, only to return 5 minutes later with a bottle of champagne. We would all toast to friendship, and I would silently ponder the term paper I had just finished for a course in the law school: "Obligations and Duties of Trustees in a California Non-Profit Corporation." What would the professor think of the scene I'd just witnessed?

I devoted most of my attention to my studies and to my duties at *Dr. Dobb's*—there wasn't time for much else—but I always managed to keep aware of what Allison and Albrecht were up to. (All of you who know them are probably saying, "How could she *help* it?") They were always trying to tell me that I'd never learn from a book what I needed to know about life and my work, and about getting along with people. For two years I argued with them, together and individually, telling them I just didn't understand the haphazard, unplanned, chaotic way they went about doing things. They kept telling me that I didn't know it yet, but I was just as crazy as they were.

I suppose they were right. In the end when I'd finished my schoolwork and learned everything I could at *Dr. Dobb' Journal*, I turned down each of the highly-paid job offers that came my way. I moved to Sausalito to starve and write a novel. There are a couple of characters in it who remind me of Allison and Albrecht; they're quite vivid, extremely bohemian and very strong. With the exception of the female protagonist, they rather steal the show.

Mac the Hack by Tom Pittman

I recently attended a "Hackers Conference," where one of the subjects of discussion was the definition of *hacker*. There was no consensus. Why?

When Apple introduced the Macintosh, I got excited about it just like everybody else. This was the computer I really wanted! At the Hackers Conference, over half the people were into the Mac; none was willing to admit to spending his time on IBM. Why?

About the time Jim Warren sold the *West Coast Computer Faire*, most of us had noticed that the fun was gone. Where were "the good old days?" Steven Levy's book *Hackers* chronicles "the last hacker." The *Homebrew Computer Club* just isn't what it used to be. Why?

When I started to write this, I thought I had answers to these questions, a sort of "unified field theory" of computerism. I'm not so sure any more, but at least I can share some ideas.

A hacker is an artist, and computer artistry is not distinguished from other art forms except in the medium chosen.

Just as paintings range from great (e.g. Rembrandt, Dali) to professional (painted houses and cars) to spray-can grafitti, so computer artisty ranges from the sublime (e.g. MacPaint, $\TeX$) to the professional (1-2-3 and MSDOS) to the so-called 414s. Unlike painting, sculpture, music, and the like, few people can really appreciate the artistry in a computer product. Perhaps that will change.

It is the nature of great art that it compresses a great deal of design, intellectual sweat and individual personality into the created object. There are no great paintings painted by a committee of artists. Curiously, it seems to matter little what tools the artist had, or where his starting point was, provided that his accomplishment from that point exceeds what the rest of us could have done. Ansel Adams had color and motion available, but he chose to limit himself to black and white stills—and what art he created! A virtuoso on a violin produces art; the same sound from a Moog is ho-hum.

There is another facet to computing that deserves exploration: the urge to produce results that are impossible. There is a feeling of accomplishment in climbing the north face of Halfdome for the first time, in breaking the ("impossible") four-minute mile or the sound barrier, in producing a tune from a computer with only 256 bytes of RAM. This same feeling can accompany feats that are impossible only because some authority said "you can't" (or must not). Thus, the urge to create finds its expression in the hacker who cracks the protection scheme in commercial software, or breaks into a "secure" computer installation with his modem.

Macintosh is a computer that says, "you can't." It is promoted as a computer that is easy to use, but the box is sealed with screws you can't reach, the software (in ROM) can't be changed, and the development tools can't be obtained. Although I don't think that is why computer people buy it, still after you get over the realization that it does not do all they seem to claim for it, you are left with a lot of "impossible" things to challenge the aspiring intellect, a lot of opportunity to show off your freedom from such oppressive forces.

The computer hacker, like any real human being, wants recognition. I write programs for a living. I don't know if that qualifies me as a hacker, but it is more than just a job: it's *fun*. But I write my best stuff for other people, or at least to help me do things for others. I never seem to get around to doing things just for me.

To get recognition requires that the program (or hardware, or whatever) be noticeably outstanding. It must be head-and-shoulders above the competition, so that it is *clear* how good it is. If you start with limited tools you can do an outstanding job, but who can tell? Start with the best tools (would you believe Mac? Next year, anyway), and then your Herculean effort will be seen for what it is. The result: recognition, praise, glory.

Computing is a drug. Some people take aspirin to cure a headache, so they can get on with what they are trying to do. Others pop pills just for the immediate effects. Drugs have a way of growing on you. Once you have free-based Unix, it's pretty hard to go back to snorting CP/M. Mac gives a high something like Unix—it would be way up in the stratosphere if it had better software tools, like Unix already has—but Mac costs a lot less than Unix: it's affordable.

Computing is a religion. Centuries ago, in the 1940's, the founders of this religion (Turing, Von Neumann, etc.)

Take your software public!

We're building a "demo data base" of available software for every retailer to access. We want to add yours to it.

Is your software the best kept secret in the industry?

The Program Finder can give it nationwide exposure! It is a unique marketing service that allows your software to be featured in a demo-library accessed by a rapidly expanding list of retailers coast to coast.

When a customer is interested in your product category, the salesperson dials into our data base using any operating computer. Together, they screen through available packages. The Program Finder then displays information about you, your software, your prices.

They are then prompted through a demo you have written, assuring that your package is properly demonstrated. Included in the demo is, of course, how to order your product.

Don't let your software sit around waiting to be discovered. If you've got it, let The Program Finder flaunt it.

The PROGRAM FINDER

```
main( )
{       printf("Call Mitch Kolesaire, Director of Sales, at\n");
        printf("Software Information Systems, Inc.\n");
        printf("Reach him at %s\n","201-882-9141");

}
```

A product of Software Information Systems, Inc.

brought salvation to the human race, entrusting the Holy Truth to a small band of devoted disciples. As these saints spread the Gospel, great churches grew up (IBM, Univac) with an ecclesiastical hierarchy (we call them "priests" even today). A few anabaptists (the hackers at MIT) nipped at their heels, but the Reformation happened in Silicon Valley, led by Martin "8080" Luther; the doctrine of universal priesthood (all believers are priests) was once again gospel.

Not for long. Slowly the encrustations of institutional religion again took over. Scientific Enlightenment (bean counters in three-piece suits) replaced the faith of the Reformation. But there is hope! John Wesley and Billy Graham sparked revival in Christendom; Macintosh is doing the same for Computerdom. First Jimmy, then Ronnie; now Woz for President!

The urge to create, the delirious feeling of power, the longing for praise: these are universal human drives, dating back to the first rejection of authority in the Garden of Eden. The first vaporware announcement was, "You shall not surely die, but you shall be like gods." Macintosh, is thy name Serpent?

Tiny Hackers: a Realizable Fantasy by Bob Albrecht (with Mike Swaine)

Ob returns.

I first used the expression "realizable fantasies" in the *PCC* newsletter back before there was a *DDJ*. Starting next issue I'll be back in the pages of this magazine doing a column called "Realizable Fantasies" with Mike Swaine because we both think that people doing things with computers today need a kick in the imagination. The idea of the column will be to open discussion each month on some project that any fool can see is a good idea, but that, for whatever reason, nobody is carrying out. A public-domain Unix, an open-architecture Macclone, things like that.

This Festschrift paper can be considered the pilot for the series, because I have such a realizable fantasy. It has a name. I call it Tiny Hackers, and like any good fantasy, it wears a different face every time I look at it. Its origin goes back to before *DDJ*, back to the early sixties when I started teaching kids to play with computers. It recently got rekindled by listening to Brian Harvey, at the Hackers' Conference you're hearing so much about in this issue, talking about passing on knowledge to a new generation of hackers.

I have been working with kids and computers since 1962. I wasn't really having a good time in life until I began teaching kids about computers. I now work with tiny kids three to six years old, helping them to play with computers, although most of them are not, strictly speaking, hackers.

My corporation, *Dragonquest*, works with the East Menlo Park Boys' Club. Boys' Clubs are an ideal place for hackers who want to bring wonderment to kids. They are open to girls as well, and there are also Girls' Clubs. If kids are going to do things with computers, they're going to do so at home or at places like the clubs, not in the schools, which are the last place you're going to see any hacking. The stuff they're doing in schools is dreadful.

Around 1981 we started Computer Kids, of which Tim Finger is the director. With Computer Kids, we're bringing such things as programs that teach touch typing into the clubs. Once somebody gets comfortable with using the machine to store and print words, it's only a short step to seeing the possibility of putting your thoughts into words and printing many copies and distributing them; and that's a newsletter. We'd like to explore the possibility of kids in the Black community getting into self-publishing and using the newsletter to find out what's going on around them.

I want to bring tools to these kids, powerful tools. Future tools. Important tools. Once they have those tools, some of them are going to become the next generation of hackers; they will be the kids who redefine the word hacker.

What kind of tools should they have? Is there a Tiny Hackers' machine? Cheap, powerful, designed so small children can use it effectively and artfully?

If you dig through back issues of *Rainbow*, a magazine for users of the Radio Shack Color Computer, you can find a wealth of tips for turning the Color Computer into a system with surprising programming power, like a cheap 128K upgrade. I priced out a system: $750 will give you a Color Computer with O/S-9, a Unix-like operating system. There is a Basic compiler available, a Pascal compiler and a C compiler. That's one possibility for a Tiny Hackers' machine. [*Radio Shack may in fact bring out a Color Computer in 1985 with 128K RAM, an RGB monitor and O/S-9 in ROM. —Ed.*]

But there are other possibilities. Now is the time for somebody to have the courage to produce a home computer with Logo built in. With sprites. Kids don't give a damn about turtle graphics, but they love sprites. And a touch pad and a keyboard with keys in alphabetical order so the kids can find the letters. The Qwerty layout is about as easy to use as if the letters were distributed randomly. Why put barriers like that in kids' way?

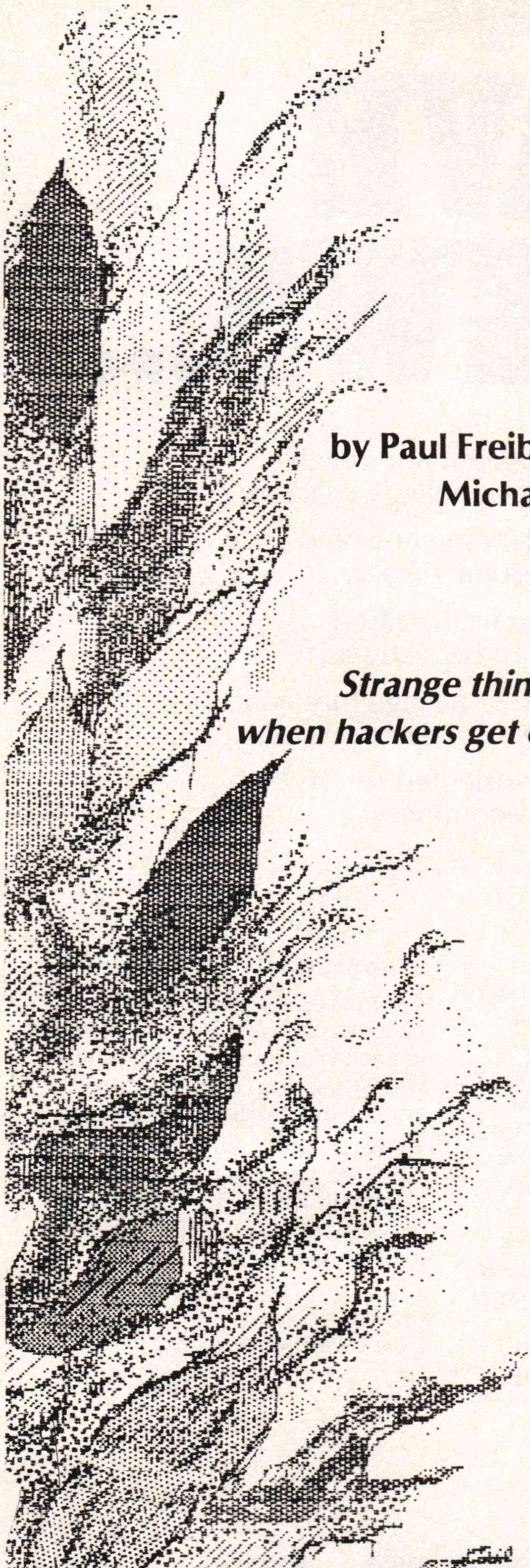
It would be great to see kids in the clubs building Lee Felsenstein's proposed Hackers' Macs from kits.

So that's the realizable fantasy: Tiny Hackers. It's a confluence of ideas that could split off into many streams: projects, columns, newsletters. One of the forgotten motivations behind the original Tiny BASIC was that it be a language for tiny *people.* Maybe we still need a tiny language. If you are interested in the next generation of hackers, if you think that we need home machines with sprite chips or that computers for kids should use random number generators that return integer values, consider *DDJ* a forum for bringing the issues forward.

I am starting a small, personal newsletter called *Dragonsmoke*, and I suspect that some of these issues will also come up there. The focus will be on whatever *Dragonquest* is up to, including the work at the Boys' Clubs. If you'd like to be on the mailing list, send an SASE to PO Box 310, Menlo Park CA 94026 (that's the original 1976 *DDJ* PO box).

But the thing I'd most like to see is for some of *DDJ*'s readers to do something good for a Boys' Club or Girls' Club. Dig out those old Sols and Imsais. Better yet, give time. **DDJ**

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 192.**



FIRE IN THE

by Paul Freiberger and Michael Swaine

Strange things happen when hackers get organized.

There was a strong feeling [at the Homebrew Club] that we were subversives. We were subverting the way the giant corporations had run things. We were upsetting the establishment, forcing our mores into the industry. I was amazed that we could continue to meet without people arriving with bayonets to arrest the lot of us.

Keith Britton

The Homebrew Computer Club

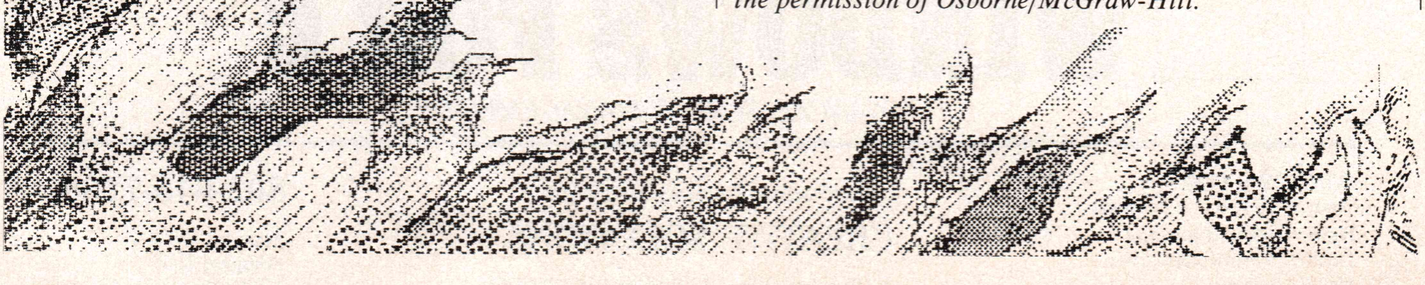
Felsenstein and Marsh weren't there during Homebrew's gestation. Early in 1975, a number of San Francisco Bay Area counterculture information exchanges existed for people interested in computers. Community Memory was one. There was also PCC [People's Computer Company] and the PCC spin-off, the Community Computer Center. In addition, peace activist Fred Moore was running a non-computerized information network out of the Whole Earth Truck Store in Menlo Park, matching people with common interests about anything, not just computers. Moore became interested in computers when he realized he needed a machine and a base of operations, and he talked to Bob Albrecht at PCC about both. Soon, Moore was teaching children about computers and learning about them himself.

At the same time, Albrecht had been looking for someone to write certain assembly language programs and found Gordon French, a mechanical engineer and computer hobbyist, who then supported himself building slot car motors.

When the Altair story appeared in *Popular Electronics*, the need for a more direct information exchange became clear. The PCC people took the Altair seriously from the beginning. Keith Britton, a demolition consultant and PCC's treasurer, thought its arrival foretold the eventual demise of the computer priesthood. "All of us were champing at the bit

---

Michael Swaine, Editor-in-Chief of DDJ
Paul Freiberger, West Coast Editor of Popular Computing
From Fire in the Valley: The Making of the Personal Computer. *Copyright 2/3 1984 by McGraw-Hill, Inc. Used with the permission of Osborne/McGraw-Hill.*



VALLEY

Photo 1
Gordon French, one of the founding members, addressing a meeting of the Homebrew Computer Club in 1979

to get an Altair," French recalls. So Fred Moore pulled out his list of the computer curious, the revolutionaries, the techies, and the educational innovators, and sent out the call. "Are you building your own computer? Terminal? TV Typewriter? I/O device? Or some other digital black box? Or are you buying time on a time-sharing service?" Moore's flyer asked. "If so, you might like to come to a gathering of people with like-minded interests. Exchange information, swap ideas, talk shop, help work on a project, whatever." The announcement tentatively called the group the Amateur Computer Users Group or Homebrew Computer Club, and it met on March 5, 1975, in Gordon French's garage.

Felsenstein read about the upcoming meeting and intended not to miss it. He collared Bob Marsh and they drove in Felsenstein's pickup truck through the rain across the Bay Bridge to the peninsula that stretches from San Francisco in the north to Silicon Valley in the south. Gordon French lived in suburban Menlo Park, a town jogging distance from Stanford and right on the edge of Silicon Valley.

At the first meeting, Steve Dompier reported on his visit to Albuquerque. MITS had shipped 1500 Altairs and expected to ship 1100 more that month. The company was staggering under the weight of the orders and couldn't begin to fill them, Dompier said. Bob Albrecht displayed the Altair that PCC had received that week—PCC was just behind Harry Garland and Roger Melen, over at Stanford, on MITS's list—and passed out the latest issue of PCC.

Dompier, like Marsh and Felsenstein, had driven down from Berkeley, but most of the 32 or so attendees were from the San Francisco Peninsula. Albrecht and Gordon French, who chaired the meeting, and Fred Moore, who took notes for a newsletter, and Bob Reiling, who soon took over that newsletter, all lived in Menlo Park. Many other people had come from farther south—from deeper into Silicon Valley: Mountain View, Sunnyvale, Cupertino, San Jose—people like Allen Baum, Steve Wozniak, and Tom Pittman, who described himself as a microcomputer consultant, perhaps the first in the world.

As the meeting concluded, one Homebrewer held up an Intel 8008 chip, and asked who could use it, and gave it away. Many people present that night sensed the opportunity in this Homebrew spirit and in Dompier's words. One of

them was Bob Marsh. Marsh went immediately to see Gary Ingram about forming a business enterprise. I've got a garage, he said. It seemed enough. They decided to call themselves Processor Technology, or Proc Tech. Marsh designed three plug-in circuit boards for the Altair: two I/O boards and a memory board. They looked good, he thought. He devised a flyer announcing Proc Tech's products, made hundreds of copies of it on a campus photocopying machine, and took 300 of them to distribute at the third meeting.

By this time the club was flourishing. Fred Moore was exchanging newsletters with Hal Singer, who put out the *Micro-8 Newsletter* in Southern California and had formed a Micro-8 club shortly after Homebrew started. Other publications were passed around. PCC and Hal Chamberlain's *The Computer Hobbyist* attracted special interest. A Denver organization identifying itself as a provider of support for Micro-8 and TV Typewriter hobbyists and calling itself The Digital Group offered subscriptions to its newsletter. The movement was getting hard to keep up with. Intel, with its 4004, 8008, and 8080 chips, and at least 15 other semiconductor manufacturers had introduced microprocessors into the market, and the newly formed club tried to keep its members informed about them all.

The third Homebrew meeting drew several hundred people, too many for Gordon French's garage. It was moved to the Coleman mansion, a Victorian building serving as a schoolhouse. There Marsh gave a brief talk, explaining that he was selling memory and I/O boards for the Altair. He hoped to present Proc Tech as a serious company, not just the fancy of an unemployed electronics engineer with access to a copying machine. He offered a 20% discount for cash prepayment. To his disappointment, no one came to talk to him at or after the meeting.

The next week the first order arrived. Garland and Melen were seeking Processor Technology's cheapest advertised product. Their request was written on the stationery of their new company, Cromemco. They sent no check, just a purchase order requesting 30 days net credit, hardly what Marsh had expected. He had made Proc Tech look like a real company, all right.

After the Cromemco order, many others followed, most enclosing cash. Ingram fronted $360 for an advertisement in *Byte.* With the cash streaming in, Marsh and Ingram could afford to advertise in *Popular Electronics,* and they did, spending $1000 for a one-sixth-page ad. They incorporated, and Ingram was made president. As corporate headquarters and factory, they had half of an 1100-square-foot garage, no products, no schematics for proposed products, no supplies, no employees, and thousands of dollars in cash orders. It was beginning to appear that they had some work ahead of them.

Meanwhile, Lee Felsenstein was getting more involved with Homebrew. He took over from Gordon French as the master of ceremonies—he refused to think of himself as a chairman. The meetings were now held in the auditorium at the Stanford Linear Accelerator Center. Over the years, Felsenstein became intimately associated with the club and fostered its anarchic structure. The group had no official membership, no dues, and was open to everyone. Its newsletter, offered free after a nudge from Felsenstein, became a pointer to information sources and a link between hobbyists. As group toastmaster, Felsenstein performed with a curious kind of populist showmanship. As one attendee, Chris Espinosa, said, "People call him the Johnny Carson of Homebrew, but he's more than that. He kept order, he kept things moving, he made it *fun* to go to the meetings. There were 750 people in that room at one time, and he worked it like a rock

Photo 2
Chuck Grant and Mark Greenberg, founders of Kentucky Fried Computers and North Star Computers, with a NorthStar Horizon

concert. It's hard to describe, but to see him work a crowd like a Baptist preacher. . . . He was great."

The meetings didn't follow Robert's *Rules of Order* with Felsenstein running them: he gave them their own special structure. First came a mapping session, during which Felsenstein recognized people who briefly proffered their interests, questions, rumors, or plans. Felsenstein sometimes had quick answers to their questions or witty comments on their plans. A formal presentation followed, generally of someone's latest invention. Finally, there was the Random Access session, in which everyone scrambled around the auditorium to meet those they felt had interests in common with them. It worked brilliantly, and numerous companies were formed. A remarkable amount of information was exchanged at those meetings, and much information had to be exchanged; they were all in unfamiliar territory.

About this time, a San Francisco branch of Homebrew started. It held its first meeting at the Lawrence Hall of Science in Berkeley. Although it was called the San Francisco branch, Berkeley was a logical place for it to meet. Universities were becoming hotbeds of self-taught microcomputer expertise. Professors with grant money now found it cost-effective to buy minicomputers rather than buy time on the university mainframe computer, which was invariably out of date and overworked. DEC was selling PDP-8 and PDP-11 minicomputers to professors as fast as it could build them. They were particularly popular in psychology labs, where they were used for experimenting on human subjects, automating rat and pigeon labs, and analyzing data. The invasion of the psych lab by minicomputers created a new kind of expert: one who might know something about psychological research, but who was more clearly a hacker and a computer nut—someone to figure out how to run the computer and make it do what professors, who were generally ignorant about the machine, wanted.

Howard Fulmer was such a person. Fulmer worked in the Psychology Department at UC Berkeley running PDP-11s, selecting minicomputers for professors to buy, building interfaces, and programming experiments. In early 1975, one of Fulmer's professors bought an Altair, and Fulmer learned to use it. Soon after, Fulmer left his job to devote more time to microcomputers.

He was not alone: the Altair raided the University of California at Berkeley. George Morrow, a graduate student in math, worked at the university's Center for Research in Management Science with two other students, Chuck Grant and Mark Greenberg. They were trying to develop a language to use with a microprocessor in computer-controlled research.

Morrow, Grant, and Greenberg found that they worked well together. All three were perfectionists, although in different ways. Morrow, thin, prematurely balding, and with a twinkle in his eye and an irrepressible wit, seemed always to be enjoying himself, perhaps especially when he was hard at work. Grant and Greenberg appeared cut from a darker cloth. They were more businesslike. Although Grant and Greenberg often attended Homebrew meetings and profited from the free, open exchange of information, they never considered themselves part of the hobbyist community. Technically, the three formed a good team. Morrow knew hardware, Grant preferred software, and Greenberg was at home with either.

Photo 3
George Morrow in a pose for an early advertisement

The trio considered making boards for the Altair or even a kit computer of their own. They knew that they were a good design team, but they also knew they lacked sophistication in marketing. So Morrow sought the advice of Bill Godbout. Middle-aged, blunt, opinionated, with a paunch that he joked about and an airplane that he flew stunts in, Godbout was the electronics distributor whom Bob Marsh had tried to interest in his walnut digital clock when he and Felsenstein had first worked in the garage. Morrow told Godbout about their plans.

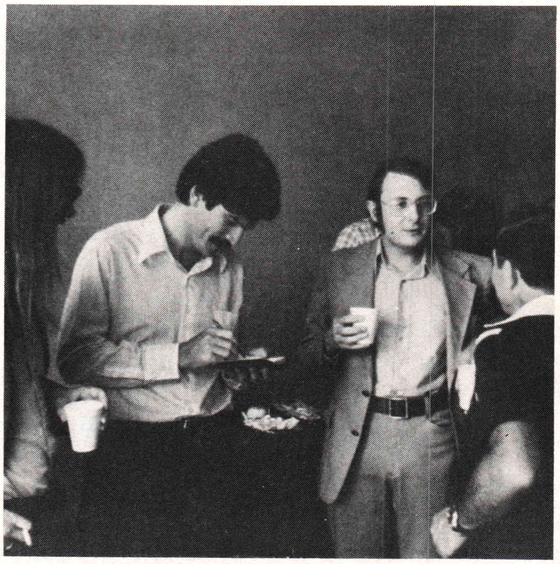


Photo 4
Steve Dompier, Bob Marsh, and Lee Felsenstein at a meeting of Processor Technology dealers in 1979.

Godbout was then selling chips and minicomputer memory boards by mail, and Morrow asked if he intended to sell Altair memory boards. Godbout scoffed. He wouldn't so dignify the product, he said. Morrow wondered if he might be interested in distributing a good computer, the creation of a top-notch design team. "You guys?" Godbout asked. He looked Morrow over. Godbout believed he was good at sizing people up, and Morrow looked all right. They agreed to split profits down the middle and shook on it. No written contract, Godbout said. Written contracts were a sign of mistrust and an invention of lawyers, and if there was anybody Godbout didn't trust, it was a lawyer.

By this time a motley group of engineers and revolutionaries had assembled in Silicon Valley in the infancy of a billion-dollar industry: irascible Bill Godbout, who suspected lawyers; ex-*Berkeley Barb* technical editor and current Homebrew toastmaster Lee Felsenstein; Bob Albrecht, who left a high-paying career to teach children about computers, who smoked cigars, and called himself "The Dragon"; Bob Marsh, testing his own abilities, turning his love for electronics into a garage corporation; and Keith Britton, who saw himself and the other Homebrewers as pivotal in "an equivalent of the industrial revolution but profoundly more important to the human race." A surprising number of them held political views that would have shocked the local Rotary Club, and almost all had no love for IBM and the computer establishment. But they and others like them were pulling off the most startling entrepreneurial achievement in recent times. And much of the action took place at Homebrew.

The Homebrew Computer Club was not merely the spawning ground of many Silicon Valley microcomputer companies. It was also the intellectual nutrient in which they first swam. Presidents of competing companies and chief engineers would gather there to argue design philosophy and announce new products. Statements made at Homebrew changed the directions of corporations. Homebrew was a respected critic of microcomputer products. The Homebrewers were sharp, and could spot shoddy merchandise and items that were difficult to maintain. They blew the whistle on faulty equipment and meted out praise for solid engineering and convivial technologies. Homebrewers soon developed the power to make or break new companies. In part due to Lee Felsenstein, Homebrew encouraged the conviction that computers should be used for and not against people. Homebrew thrived in a kind of joyous anarchy, but the club was also an important step in the development of a multi-billion dollar industry. Processor Technology was one of the children of Homebrew.

The first part of the meeting we were involved in open combat with Intel. Intel was out to torpedo any standardization effort on the S-100 bus.

George Morrow

Home Rule

A continuing fear in the developing microcomputing industry was that "the big boys" would come in and spoil all the fun. Sometimes "the big boys" meant IBM and the other mainframe computer and minicomputer companies; sometimes it meant Texas Instruments, Commodore, and the other electronics companies that had waged Pyrrhic price-cutting wars in the calculator industry; especially it meant Texas Instruments, known for its ruthless price-cutting. Lee Felsenstein summarized the dread of the hobbyist entrepreneurs: "Anyone but TI!" Intel and some of the other semiconductor companies, although well situated to produce microcomputers from their own chips, had expressed reluctance to do anything that could be construed as competing with their own customers, and the hobby-born microcomputer companies had developed enough clout by this time to be taken seriously as semiconductor customers.

In December of 1976, Commodore International, an electronics firm with a lot of market muscle, leaked information to *Electronic Engineering Times* about a new product. Commodore, the story went, was ready to release a machine very much like a low-cost Sol. Proc Tech was just shipping the first Sols and Marsh was thinking about the company's next product, a new version of the Sol with an integrated keyboard and 64K of memory, that would be cheap at $1000. It was, unfortunately, essentially the Commodore machine.

Convinced that Commodore actually had the computer on the launch pad and that Proc Tech could never compete with it, and even more worried by news that National Semiconductor was also planning a microcomputer, Marsh scrapped the project. The laws of battle in the semiconductor wars five years earlier had been to cut prices to the baseline and push the technology madly, even under threat of corporate extinction. Proc Tech couldn't compete with National, especially in mortal combat. But in fact the Commodore machine would not appear for some time, and the National computer never materialized at all.

Many new hobbyist-born companies were starting to manufacture microcomputer products, but most of these were turning out boards for the Altair or IMSAI, and practically all were small companies, start-ups like Proc Tech.

Howard Fulmer began such a firm in his Oakland basement. After reading an editorial by Ed Roberts in Dave Bunnell's *Computer Notes* that attacked the compatible board companies as "parasites," he considered calling his own company Symbiotic Engineering to emphasize his conception of the proper relationship between MITS's products and his own. But a group called the Symbionese Liberation Army was making a name for itself then, and he wanted to avoid confusion. He called his company Parasitic Engineering.

George Morrow and Howard Fulmer were both designing Altair-compatible products, and in the spring of 1977 decided to build a computer together. Morrow would supply Fulmer the boards he had designed at a cheap price, and Fulmer would devise the remainder of the computer. Fulmer called it the Equinox 100. It was a solid design, for they had listened to the ideas of Bob Mullen, one of the founders of Diablo Systems, a Silicon Valley disk-drive manufacturer, and of Bill Godbout about improving the S-100 bus.

The timing of the machine's release was unfortunate though. The Equinox was an 8080 machine, and Technical Design Labs in New Jersey, Garland and Melen's Cromemco, and The Digital Group in Denver were all known to be designing computers based on the new and apparently better Z80 chip. Cromemco had already produced a Z80

central processor board, and hobbyists were dropping it into the IMSAI chassis to create a mongrel Z80 machine.

Marsh wondered if Proc Tech shouldn't do a Z80 machine as well. But it seemed irrational to dump a successful design to achieve a marginal improvement in performance. Besides, the processor mattered much less than the software, he believed. The software made the computer work, and that would distinguish one machine from another.

Proc Tech called two programmers, Jerry Kirk and Paul Greenfield of MicroTech in Sunnyvale, who had produced high-level language compilers for minicomputers. They were hired to create a set of programmer's tools, programs that would make it easier to write, edit, and debug other programs on the Sol. Ingram developed their work into Software Package One.

Ownership of software was an inflammatory issue in the Valley and elsewhere. Proc Tech was aggressively pro-piracy, and its hobbyist founders swapped program tapes at Homebrew meetings along with everyone else. Gordon French, who after helping to start Homebrew had become Proc Tech's General Factotum (his official title), argued for an open system, that is, free dissemination of software code and internal workings to everyone. He wanted outside programmers and peripheral manufacturers to be able to create compatible products and expand the market. At that time, Ed Roberts and the entire mainframe and minicomputer industry held the opposite view. But the hobbyists were bringing their own values into their industry. An open architecture, the publicly known physical design of the machine, was one emerging ideal. An open operating system was another. But at Proc Tech the idea of an open operating system was frowned upon. Marsh and Ingram wanted a proprietary operating system.

In fact, Proc Tech had its own disk operating system very early on. The company bought PTDOS from its author, 19-year-old Bill Levy, who developed it at the Lawrence Hall of Science at Berkeley. Levy modeled PTDOS after Unix, a mainframe/minicomputer operating system in use at UC Berkeley. Marsh thought PTDOS much better than CP/M, but PTDOS was slow to reach the market because of "the drive fiasco."

Disk drives posed an alluring challenge in 1976, when the Sol was released. They existed and were used heavily in mainframe and minicomputers, but to mount a disk drive on a microcomputer was prohibitively expensive. Drives typically cost $3500. So Marsh was very intrigued when George Comstock, Bob Mullen's partner at Diablo Systems, declared at Homebrew one night that he wanted to develop a disk drive for microcomputers. Comstock thought that a drive, complete with a controller board and software, could be sold for about $1000.

But Diablo was not then involved in the growing microcomputer industry, and Comstock felt a disk-drive system would only flail around without close consultation with microcomputer companies. He later proposed a joint effort to Marsh. Diablo would develop the drives, the physical mechanisms that read and write information from and to disks, and Processor Technology would write the software and develop an S-100 board to control the drives. He also proposed that Proc Tech could market the board on its own.

Disk drives were so clearly destined to belong in any serious microcomputer system that engineers were already vying to develop a low-cost drive system with software and a controller board. Shugart's 5¼-inch drives seemed attractive, but they had one drawback. IBM had been using 8-inch drives and had established certain standards for the devices. There were no standards for small disk drives and no guarantees that disks written on one brand of machine would be readable on another. North Star had selected the Shugart drive and sold it for under $800. Using an idea of Eugene Fisher's of Lawrence Livermore Labs, both Morrow and San Francisco engineer Ben Cooper had begun developing relatively low-cost 8-inch disk drives. Cooper had perhaps the first commercial 8-inch disk controller for microcomputers. Morrow, shortly thereafter, had the first one available for the $1000 price Comstock was aiming for, and he negotiated with Digital Research and Microsoft for an operating system (CP/M) and BASIC to distribute free with the drive system. Both Morrow and Cooper continued to develop significant disk products, and Cooper created the first hard disk controller for microcomputers.

But at Proc Tech, the disk drive plans were crumbling. Diablo encountered trouble with the drives and dropped the project, leaving Proc Tech so far into development of the controller that it had to continue. Marsh and Ingram raised the price of the system to $1700 and substituted a more expensive drive offered by Perscii. The price was too high, and Proc Tech's drives didn't always work. Customers could find better deals from Cooper, Morrow, and North Star.

Despite such problems, Proc Tech seemed to be thriving. The executives were recycling their profits into the company. (Lee Felsenstein was investing his in the Community Memory project.) The Proc Tech staff in Emeryville now numbered 85, not counting non-employee/consultant Felsenstein, and headquarters was growing crowded. That year, 1977, Proc Tech moved south to the bedroom community of Pleasanton. The new offices boasted a spacious executive suite with large windows looking out over the valley.

But there was competition. As 1977 came to an end, Proc Tech found itself in a more serious industry. The trading of information, the shirt-sleeve management, the flashes of idealism, and the lack of detailed planning that had characterized the industry from the start still existed. The chief users and the designers and company presidents were still hobbyists at heart, and most of the world knew nothing of the revolution that was afoot. But new companies were emerging like mushrooms overnight. Among the computer and computer-related companies at the end of 1977 were Apple (which some insiders thought had great promise), Exidy, IMSAI, Digital Microsystems, Alpha Micro Systems, Commodore, Midwest Scientific, GNAT, Southwest Technical Products, MITS, Technical Design Labs, Vector Graphic, Ithaca Audio, Heathkit, Cromemco, MOS Technology, RCA, TEI, Ohio Scientific, The Digital Group, Micromation, Polymorphic Systems, Parasitic Engineering, Godbout Engineering, Radio Shack, Dynabyte, North Star, Morrow's Microstuff, and, of course, Processor Technology.

Many companies were located in the Bay Area and were associated with the Homebrew Club. The club had become large, and by 1977 tended to assemble in fairly predictable

Circle **no. 10** on reader service card.

groups. In front, performing, was Lee Felsenstein. Bob Marsh and the Proc Tech group were usually assembled along one wall. Steve Wozniak and the boys from Apple and the other 6502 processor fans sat in the back. Jim Warren of *Dr. Dobb's* sat on the aisle three seats from the back, stage left, ready to stand during the mapping session and do his Core Dump, an extemporaneous outpouring of all the news and rumors he had heard. The front row always had Gordon French, who maintained the software library, and Bob Reiling, who wrote the newsletter.

In December 1977, Reiling wrote, "The development of special-interest groups has probably been the biggest change during the past year. At the beginning of the year the 6800 group was holding regular meetings. At the end of 1977, the groups include not only the 6800 group, but also the F8 Users, North Star Users Group, Sol Users Society, and PET Users." At that time, the Homebrew attendees (the club did not have members) included key people from Apple, Cromemco, Commodore, Computer Faire, *Dr. Dobb's,* Itty Bitty Computers, M&R Enterprises, Mountain Hardware, IBEX, Mullen Computer Boards, North Star, PCC, Proc Tech, and the Bay Area computer stores. The most significant of these then was Proc Tech. Marsh had, to some extent, realized his dream. The company seemed to be doing very well.

And in December, Reiling could report optimistically, "The IEEE now has a standards group to sort out the various hardware and software standards." That blithe statement subsumed a wrangling struggle and a remarkable achievement which brought new legitimacy to the industry. The sorting out had been no simple matter.

Bob Stewart was a consultant in optics and electronics and a member of the Institute for Electrical and Electronics Engineers (IEEE). He had bought an Altair and had become frustrated with it. At a meeting at Diablo Valley College to discuss the S-100 bus he met some company presidents: Harry Garland of Cromemco, Howard Fulmer of Parasitic Engineering, Ben Cooper of Micromation, and George Morrow of what he was then calling Thinkertoys. *Byte*'s Carl Helmers was also there. The idea was to cure the obvious problems of the bus and to establish common standards, so that one company's board would work with another's. Garland explained the virtues of his and Melen's shielded bus, but Morrow thought he had a better approach. No immediate agreement was forthcoming. Stewart suggested creating an official IEEE standard for the bus. With the group's encouragement, he petitioned the IEEE to form a microcomputer standards subcommittee of the computer standards committee. The petition succeeded, and the group became official.

Roberts was invited to participate in the microcomputer standards subcommittee, but declined to send a representative or even to respond directly. He did say in print that he felt MITS had the sole right to define the bus. The subcommittee ignored him. At first, the meetings involved contention with Intel, which fought standardization. Morrow got the impression that Intel wanted no standards unless Intel was setting them. But when the subcommittee decided to formulate standards whether Intel liked them or not, Intel acquiesced. This was outrageous cheek. A bunch of hobbyists turned entrepreneurs had simply ignored the biggest microcomputer company of that time and had faced the leading chip manufacturer and not been struck by lightning.

In spite of its solidarity, the subcommittee had no guarantee that it could really create standards. The subcommittee had 15 assertive, opinionated people disputing an issue about which they held legitimate and conceivably irresolvable differences. Each of the members had a product that would be incompatible with anything likely to be proposed. As the meetings went on, Roger Melen came in for Cromemco. Alpha Micro was represented. Elwood Douglas appeared for Proc Tech and judged the standard against the memory board he was designing. George Millard spoke for North Star. Someone arrived from IMSAI to read its formal position, which resembled Ed Roberts'. The subcommittee ignored that position too. Most of its members had written IMSAI off as a place where training in *est* mattered more than training in engineering.

At times the subcommittee members weren't too fond of each other. They argued for hours, with no one yielding an inch. They would then return to their companies and discuss how to compromise their own designs to achieve a standard. At the next meeting, they would find themselves closer to agreement. Little by little, these creative, independent people subordinated their egos and short-term economic gains for the good of the entire microcomputer field.

The committee was attempting guerrilla design. In mainframes and minicomputers, the bus was always whatever the bus designer said it was. Although the IEEE suggested subtle variations in tolerance during the process of formalizing the company bus into a standard, independent committees did not assemble to redesign the whole bus. Timing parameters and other features were dictated by the companies. IBM and DEC worked this way. In a way their method was certainly easier than communal design. But the S-100 committee members dug into the Roberts bus, figured out how it worked, and were scrapping it in favor of a new, independent bus open to all. This was a populist revolt against the tyranny of the big company, with MITS hoisted as a poor but adequate symbol of the big company. The revolution was succeeding.

DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 193**.

B. G. MICRO

P. O. Box 280298 Dallas, Texas 75228
(214) 271-5546

MasterCard VISA American Express Card

74LS

LS00	.20	LS162	.65
LS01	.20	LS163	.45
LS02	.20	LS164	.65
LS03	.20	LS165	.90
LS04	.20	LS166	.99
LS05	.20	LS169	1.25
LS08	.20	LS174	.45
LS09	.20	LS175	.40
LS10	.20	LS181	1.50
LS11	.20	LS191	.90
LS12	.35	LS192	.80
LS13	.40	LS193	.80
LS14	.40	LS194	.65
LS15	.32	LS195	.60
LS20	.20	LS196	.70
LS21	.25	LS197	.85
LS27	.28	LS221	.65
LS30	.20	LS240	1.00
LS32	.25	LS241	.80
LS33	.40	LS242	1.00
LS37	.33	LS243	1.00
LS38	.30	LS244	1.00
LS42	.40	LS245	1.00
LS51	.24	LS251	.50
LS54	.25	LS253	.50
LS55	.24	LS257	.50
LS73	.35	LS258	.55
LS74	.30	LS259	2.00
LS85	.60	LS260	.50
LS86	.25	LS266	.40
LS90	.50	LS273	1.00
LS93	.55	LS279	.40
LS97	2.00	LS280	1.00
LS107	.37	LS283	.50
LS109	.25	LS290	.85
LS112	.30	LS293	.85
LS122	.45	LS298	.75
LS123	.55	LS299	1.60
LS124	2.75	LS323	2.60
LS125	.40	LS348	.75
LS126	.49	LS364	1.10
LS132	.50	LS366	.45
LS133	.35	LS367	.50
LS138	.45	LS368	.40
LS139	.40	LS373	1.00
LS151	.45	LS374	1.00
LS153	.50	LS375	.50
LS154	1.20	LS377	1.00
LS155	.50	LS378	.85
LS156	.50	LS390	1.00
LS157	.40	LS393	1.00
LS158	.50	LS399	1.25
LS160	.65	LS670	1.50
LS161	.50	25LS2569	3.00

TTL

7400	.22	74121	.27
7402	.22	74123	.40
7408	.24	74125	.49
7410	.19	74151	.50
7413	.33	74153	.50
7420	.22	74154	1.19
7425	.25	74157	.50
7427	.25	74160	.80
7428	.25	74162	.60
7432	.27	74163	.60
7433	.25	74164	.80
7440	.19	74165	.80
7442	.40	74166	1.00
7451	.20	74173	.75
7473	.34	74174	.85
7474	.40	74175	.80
7483	.45	74185	1.70
7485	.55	74192	.70
7486	.30	74193	.80
7490	.35	74195	.80
7493	.33	74273	1.75
7496	.40	74367	.60
74107	.28	74390	1.40
74109	.45		

EPROM

2708 1KX8 450 n.s.	2.20
2758 1KX8 +5V 450 n.s	2.50
2716 2KX8 450 n.s.	3.20
2716-1 2KX8 350 n.s.	4.95
2732 4KX8 450 n.s.	4.00
2732A-3	5.70
2732A-35	5.20
2532 4KX8 450 n.s.	3.00
2764-25	6.00
2764-3	5.00
27128-25	12.50
27128-3	11.50
27128-45	10.00

EPROM SPECIAL

We bought a large quantity of 2708s from a computer manufacturer who redesigned their boards. We removed them from sockets, erased and verified them, and now we offer the savings to you. Complete satisfaction guaranteed.

2708
$1.49 or 10/$12.00

STATIC RAM

2016-2KX8 200 n.s.	8/24.95
2101-1 - 256X4 500 n.s.	.75
21L02-1 350 n.s.	.65
2102AL-4 L.P. 450 n.s.	.49
2111-1 256X4 500 n.s.	2.00
2114L-3 1KX4 300 n.s.	1.50 8/10.00
2125A-2 1KX1 70 n.s.	2.20
2142-3 1KX4 300 n.s.	1.50
TMS4044 (MCM6641 C-25) 4KX1 250 n.s.	8/6.00
5101 - 256X4 - CMOS	$1.00
6116P-4-2KX8 200 n.s. CMOS	8/29.95
6501-5 256X4 - CMOS	1.00

4K STATIC RAMS

LESS THAN 50¢ EACH
MK4104J-4 - 250 N.S. 18 Pin Ceramic Computer Mfg. Surplus. PRIME. Fully Static. Easy to Use. Has Same Pin Out as TMS4044, but slightly different timing. With Specs. (Mostek)

8 for 5.00 32 for 15.95
VERY LOW POWER!

DYNAMIC RAM

2108-4 8KX1	1.50
2118-4 16KX1-5Volt	1.50
4027-4KX1-250 n.s.	.80
4116-16KX1-250 n.s.	8/8.00
4116-16KX1-200 n.s.	8/11.50
4116 16KX1-150 n.s.	8/12.95
4164- +5v 64K 200 n.s.	8/32.00
4164 150 n.s.	8/34.00
TMS4416-16KX4-150 n.s.	5.25
MK4516-15 16KX1-5Volt	1.50
5280N-5 (2107B-4 • TMS4060) 4KX1	8/3.95
41256 150 n.s.	8/165.00

SPECIAL

AY3-8910
W/60 Page Manual
New Price — $7.00

5832 Clock-Calendar 2.95

8000

Z8002	20.00	8085	5.95
8035	4.50	8086-2	24.95
8039	4.50	8087-3	159.00
8080A	1.25	8088	15.00

8200

8202A	10.00	8255-5	5.00
D8203-1	29.95	8257	6.00
8212	1.50	8259A	3.50
8214	2.00	8259C-5	5.00
8216	1.75	8275	19.95
8228	3.25	8284	3.20
8237-5	7.50	8287	5.75
8250B	9.95	8288	7.50
8251	4.20		

Z80

Z80 2.5 MHZ CPU	1.25
Z80CTC	1.25
Z80DMA-DMA	7.50
Z80PIO	1.49
Z80SIO/O	3.99
Z80A-4MHZ CPU	1.99
Z80A-CTC	1.99
Z80A DART	7.50
Z80A-DMA	9.95
Z80A-PIO	1.99
Z80A SIO/O	5.99
Z80B 6 MHZ CPU	9.50

F.D. CONTROLLERS

1771 Single Density	12.95
1791 Double Density	20.00
1793	17.50
1797	20.00
2793	22.95

CONTROLLER SET

THREE CHIP SET

1797 or 1793, 2143-03, 1691

by W.D. Compare at up to 86.85.

B.G. SPECIAL

All 3 for only $22.95

UART

TR1602B (COM 2017)	1.75
IM6402-(1863)+5v High speed AY5-1013 pin out	2.95
INS 8250B	9.95

6800

6800	2.50	6840	10.00
6802	3.50	6845P	7.50
6803	5.00	6845S	7.50
6809EP	3.99	6850	2.60
6810	2.00	68A09EP	4.99
6820	3.25	68A21	3.00
6821	2.00	68B45	10.00

6500

6502	2.60	6545	5.00
6522	6.95	6551	5.00

SPECIALS

1408L8	2.00
BR1941L	5.95
LM311	.40
LF353	.60
LF356	.60
TL494	2.50
LM555	.30
733	.60
9401	5.00
DS8835	1.50
MC4024P	2.50
NE592	1.50
LM319	1.00
MC1350	1.00
LM2917	1.50
LM339	.50

SOCKETS

Low Profile SOLDER TAIL

6 Pin	14/1.00
8 Pin	13/1.00
14 Pin	10/1.00
16 Pin	8/1.00
18 Pin	8/1.00
20 Pin	7/1.00
22 Pin	7/1.00
24 Pin	6/1.00
28 Pin	6/1.00
40 Pin	5/1.00

BUY $10
GET $1.00 - FREE CHOICE

2114 SPECIAL!

COMPUTER MANUFACTURERS EXCESS INVENTORY SALE!
PRIME! 2114-300 n.s.
INCREDIBLE PRICE!

YOU SAVE!
8/$8.00

GUARANTEED

CRYSTALS

32.768 Khz SPECIAL	.65
262.144	.75
300.000	1.00
307.200	1.25
1.5432 Mhz	.75
1.8432	2.49
2.000000	2.49
2.560	1.49
3.000	1.15
3.120	1.20
3.2	1.49
3.4560	1.49
3.579545	.75
4.000	2.49
4.194304	1.50
4.433618	.75
4444.000	1.25
4.9152	2.49
4.916 Bd. Rate	1.25
5.000000	1.50
5.0688	3.75
5.616	1.59
6.000	2.49
6.176	1.49
7.164112	1.00
7.3728	1.49
8.000	1.49
9.000	1.49
9.90000	1.25
10.69425	3.75
10.8864	1.49
10.920	1.49
11.088	1.59
12.000	2.75
13.440	1.00
14.31818	2.00
15.2	1.10
16.00000	1.50
16.5888	1.49
17.430	2.49
18.2259	1.00
18.4320	1.49
20.000	3.75
21.87108	1.00
22.092	1.00
32.000	2.49
40.000	2.00
87.3333	1.00
91.000	1.00
104.8	1.00

TERMS: (Unless specified elsewhere) Add $1.50 postage, we pay balance. Orders over $50.00 add 85¢ for insurance. No C.O.D. Texas Res. add 6-1/8% Tax. 90 Day Money Back Guarantee on all items. All items subject to prior sale. Prices subject to change without notice. Foreign order - US funds only. We cannot ship to Mexico. Countries other than Canada, add $3.50 shipping and handling.

Circle no. 13 on reader service card.

Tiny BASIC for the 68000

by Gordon Brandly

Dr. Dobb's Journal was created by Dennis Allison and Bob Albrecht in 1975 as a vehicle for getting public-domain versions of the BASIC programming language into the hands of computer enthusiasts. Except for Gates and Allen's BASIC for the MITS Altair, *there was no high-level language available for the new microcomputers in 1975, only arcane assembly language, so Allison and Albrecht, interested in giving kids and tinkerers and as many people as possible access to the technology, set about developing a microcomputer BASIC and publishing reports on its development in their newsletter, PCC. Since at that time 4K of memory was a lot, they made their BASIC* Tiny.

BASIC itself is ubiquitous now and still lacking any practical standard definition; no longer Tiny, it is now Better or Professional or True. DDJ *publishes little BASIC code now, but over the years* DDJ *has continued to publish programming tools, including small versions of the C and Ada languages.*

But memory is *cheap today; does it make any sense to write about Tiny Basic for the Motorola 68000 microprocessor? The answer, we think, is yes. Tiny BASIC was never intended to be a development language, but it* is *a programming language, and it is tiny, and "cheap" is a relative term. The question is, how many people are interested in getting into the 68000 as cheaply as possible? For them, this.—Editor.*

This 3K interpreter beats Applesoft on the Sieve benchmark.

Tiny BASIC was more popular than its inventors expected and they soon found that they needed a separate newsletter just to publish Tiny BASIC developments. Dick Whipple and John Arnold of Tyler, Texas, wrote a 2.9K Tiny BASIC Extended, which was published in octal in the first issue of Dr. Dobb's Journal of Tiny *BASIC* Calisthenics and Orthodontia (Running Light without Overbyte), *in January, 1976. Other Tiny BASICs followed.*

Well, a lot followed, in fact. Virtually the whole personal computer revolution to date: the demise of the early microcomputer companies (MITS, Imsai, Proc Tech, Sphere), Apple's rise and Osborne's rise and fall, the entries of Tandy and IBM, the irony of Atari, expensive advertising and cheap memory.

Gordon Brandly, R. R. 2, Fort Sask., AB, Canada, T8L 2N8.

Remember the good old days? When the 8080 microprocessor reigned supreme, 8K of memory cost an arm and a leg, ah yes . . . Well, the years went by, microcomputers got bigger, software grew more sophisticated, and prices went up. This is just fine, of course, if you can afford the higher prices. The less fortunate among us, however, must build or buy smaller 16-bit "educational" systems. This is fine too—if you don't mind having hardly any software.

This is just the sort of situation that gave rise to *Dr. Dobb's Journal* in the "good old days." The solution back then was to publish a tiny BASIC interpreter that could be adapted to just about any 8080 microcomputer around. This solution worked fabulously and gave many a hobby computer its first taste of useful software. Well, if the solution worked once, why not again? I therefore decided to produce a tiny BA-

SIC interpreter for the relatively small 68000 systems such as the Motorola Education Computer Board, the EMS 68000 board, and so on.

To produce this BASIC, I took one of the most successful 8080 tiny BASICs, Li Chen Wang's Palo Alto Tiny BASIC (*Dr. Dobb's Journal*, May 1976), and translated it into 68000 code. I then added a few features and optimized the code a little, producing a surprisingly usable interpreter.

First, I'll describe the differences between my interpreter, Palo Alto Tiny BASIC, and the ubiquitous Microsoft BASICs. I then will describe how you can install this software on your 68000 system. Finally, I'll give my evaluation of the interpreter's present performance and how it can be improved.

Features

Those who know the original Palo Alto Tiny BASIC (or the Sherry Brothers' version on CP/M User's Group Volume 11) will find this interpreter very similar. I have made two or three changes to the interpreter's syntax to bring it closer to the *de facto* Microsoft standard. The colon is used instead of the semicolon to separate multiple statements on a line. The inequality operator (#) has been changed to the more standard < > . I also added the PEEK, POKE, CALL, BYE, LOAD, and SAVE commands, which are described later.

Those of you used to a bigger BASIC, such as the various Microsoft interpreters, will find that this version works almost the same within its limitations. Following are some excerpts from Li Chen Wang's original documentation, mixed with descriptions of my extensions.

The Language

Numbers

In this version of Tiny BASIC, all numbers are 32-bit integers and must be in the range 2,147,483,647 to −2,147,483,648. I decided to use 32 bits so that the PEEK and POKE commands could access the entire address range of the 68000. This slows down arithmetic operations somewhat, but sticking to 16 bits would have produced unnecessary complications.

Variables

There are 26 variables, denoted by the letters A through Z, and a single array @(I). The dimension of this array (i.e., the range of value of the index I) is set automatically to make use of all the memory space that is left unused by the program (i.e., 0 through SIZE/4, see the SIZE function below). All variables and array elements are four bytes long.

Functions

There are four functions:

(1) ABS(X) gives the absolute value of X.

(2) RND(X) gives a random number between 1 and X (inclusive).

(3) SIZE gives the number of bytes left unused by the program.

(4) PEEK(X) gives the value of the byte at memory location X.

Commands

The LET command

```
LET A=234-5*6, A=A/2,
    X=A-100, @(X+9)=A-1
```

will set the variable A to the value of the expression 234−5*6 (or 204), set the variable A (again) to the value of the expression A/2 (or 102), set the variable X to the value of the expression A−100 (or 2), and then set the variable @(11) to 101 (where 11 is the value of the expression X+9 and 101 is the value of the expression A−1).

The PRINT command

```
PRINT A*3+1,
    "abc 123 !@#", 'cba '
```

will print the value of the expression A*3+1 (or 307), the string of characters abc 123 !@# and the string cba, and then a CR-LF (carriage return and line feed). Note that you can use either single or double quotes to quote strings, but pairs must match. If a comma appears at the end of the PRINT command, the final CR-LF will not be printed. Note also that commas separate adjacent items (most other BASICs use the semicolon to perform this function).

```
PRINT A, B, #3, C, D, E, #10, F, G
```

will print the values of A and B in 11 spaces; the values of C, D, and E in 3 spaces; and the values of F and G in 10 spaces. The value will be printed in full even if there aren't enough spaces specified for it.

```
PRINT 'abc',_,'xxx'
```

will print the string abc, a CR without

a LF, and then the string xxx (over the abc), followed by a CR-LF.

The INPUT command

INPUT A, B

will cause Tiny BASIC to print A: and wait to read in an expression from the console. The variable A will be set to the value of this expression. Then B: will be printed and variable B set to the value of the next expression entered. Note that you can enter complete expressions as well as numbers. This enables an interesting trick: you can set the variable Y to an unusual value (e.g., 9999) and use it to get the answer to a yes-or-no question, such as:

```
10 Y=9999
    : INPUT 'Are you sleepy?'A
    : IF A=Y GOTO 100
```

The user can answer the question with the expression Y, which puts the numeric value of Y into the A variable.

```
INPUT 'What is the weight'A,
    "and size"B
```

is the same as the first INPUT example except that the prompt A: is replaced by "What is the weight:" and the prompt B: is replaced by "and size:". Again, you can use both single and double quotes as long as they match.

```
INPUT A, 'string',_,
    "another string", B
```

with the strings and the _, has the same effect as in PRINT.

The POKE command

POKE 4000+X,Y

puts the value produced by expression Y into the byte memory location specified by the expression 4000+X.

The CALL command

CALL *x*

will call a machine language subroutine at the address specified by the expression x. All of the CPU's registers except the stack pointer can be used in the subroutine.

The BYE command will return control to the resident monitor program or operating system.

The SAVE command will save your BASIC program on the storage device you provide. Details on installing this device are given in the source code. As set up for the Educational Computer Board, this command will send the program out to the host computer in an easily stored text form. This isn't, however, human-readable program text because the line numbers are represented in hexadecimal.

The LOAD command will delete the program in memory and load in a program from your storage device.

Stopping Program Execution

You can stop the execution of the program or listing of the program by pressing the control-C key on the console. Additionally, you can pause in a program listing by pressing control-S and then pressing any key to continue.

Abbreviations and Blanks

You may use blanks freely within a program except that numbers, command keywords, and function names cannot have embedded blanks.

You may abbreviate all command keywords and function names, following each by a period. For instance, P., PR., PRI., and PRIN. all stand for PRINT. You may also omit the word "LET" in LET commands. The shortest abbreviations for all the keywords are given in the Table (page 46).

Note that, in some cases, the same abbreviation applies to different keywords. The interpreter is "smart" enough to identify the correct keyword for a particular situation. For instance, if the abbreviation P. appears at the beginning of a line, it can only mean PRINT. In a statement like A=P.(8), the P. only makes sense if it stands for PEEK.

Error Reports

There are only three error conditions in Tiny BASIC. The line containing the error is printed out with a question mark inserted at the point where the error is detected.

(1) "What?" indicates an error in a statement's syntax:

```
What?
260 LET A=B+3,
    C=(3+4?. X=4
```

(2) "How?" means that the statement in question is syntactically correct, but for some reason the command can't be carried out:

```
How?
310 LET A=B*C?+2
    where B*C is larger than
       2147483647
How?
380 GOTO 412?
    where line 412 does not exist
```

(3) "Sorry." means that the interpreter understands the statement and knows how to do it but lacks sufficient memory to accomplish the task.

Error Corrections

If you notice an error in your entry before you press RETURN, you can delete characters with the backspace (control-H) key or delete the entire line with control-X. To delete an existing program line, just type the line number and press RETURN.

Installation

Now, how do you get this wonderful piece of software running on your computer? Very easily, if you have a setup similar to mine. Installation on other systems should also be fairly easy if you have access to a 68000 assembler of some kind.

My setup is a Motorola MEX-68KECB Educational Computer Board (ECB) connected between my terminal and my CP/M system. The source code was assembled with the Quelo version 1.9 public domain 68000 cross-assembler for CP/M. (By the way, if you use this assembler, you will get 36 "trim16 address" errors, which is normal.) Tiny BASIC is then loaded into the ECB and executed at the cold start address of hex 900.

BASIC programs are saved and loaded by setting up an appropriate CP/M command before using SAVE or LOAD. For example (user input is underlined):

After a program is written,
  exit to the monitor:
  > <u>BYE</u>
Enter transparent mode:
  TUTOR 1.x> <u>TM</u>
Issue a PIP command to the CP/M host:
  A> <u>PIP PROGRAM.BAS=CON:</u>
Exit transparent mode and do a BASIC warm start:
  TUTOR 1.x> <u>GO 904</u>
Do the actual save:
  <u>SAVE</u>

The warm start is an entry point into the interpreter that will preserve any program you may have already entered.

Program LOADs are done similarly, except instead of PIP you must run a small program that will wait to receive a carriage return before sending the program to the ECB. Here is a sample program in Microsoft BASIC:

DIGITAL RESEARCH COMPUTERS

(214) 225-2309

STB Systems, Incorporated

IBM PC ADD ON BOARDS

NOW IN STOCK!

RIO PLUS II™ — Multi-function board with 2 RS232 serial ports (1 standard, 1 optional), parallel I/O port, game port and clock/calendar. Comes with 64K memory (expandable to 384K) and PC Accelerator.
STB-RIO + II 64 List $395 **$286.00**

GRAPHIX PLUS II™
Multi-function video board featuring monochrome text, full screen monochrome graphics, RBG color, composite b&w video, a parallel port, a light pen interface, and an upgradable clock option.
STB-G. X List $495 **$363.00**

SUPER RIO™
RAM-I/O multi-function board with 64K memory (upgrad-able to 768K), two serial ports, parallel printer port, game port and clock/calendar.
STB-SRIO-64 List $419 **$294.00**

All above boards come with "PC Accelerator" which is a RAM Disk Emulator and Printer Spooler that vastly increases your processing speed. All boards fully assembled and tested, and are warranted by the manufacturer for 1 year!

SPECIAL OFFER: Buy any two or more of the above boards and take an additional $10 per board discount

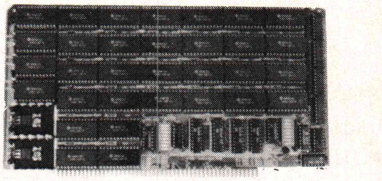
64K S100 STATIC RAM

$179.00 KIT

NEW!

LOW POWER!

150 NS ADD $10



BLANK PC BOARD WITH DOCUMENTATION
$49.95

SUPPORT ICs + CAPS
$17.50

FULL SOCKET SET
$14.50

FULLY SUPPORTS THE NEW IEEE 696 S100 STANDARD (AS PROPOSED)

FOR 56K KIT $165

ASSEMBLED AND TESTED ADD $50

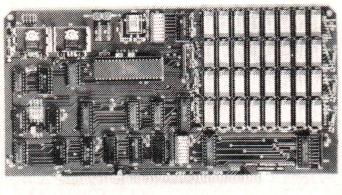
FEATURES: ***PRICE CUT!***

* Uses new 2K x 8 (TMM 2016 or HM 6116) RAMs.
* Fully supports IEEE 696 24 BIT Extended Addressing.
* 64K draws only approximately 500 MA.
* 200 NS RAMs are standard. (TOSHIBA makes TMM 2016s as fast as 100 NS. FOR YOUR HIGH SPEED APPLICATIONS.)
* SUPPORTS PHANTOM (BOTH LOWER 32K AND ENTIRE BOARD).
* 2716 EPROMs may be installed in any of top 48K.
* Any of the top 8K (E000 H AND ABOVE) may be disabled to provide windows to eliminate any possible conflicts with your system monitor, disk controller, etc.
* Perfect for small systems since BOTH RAM and EPROM may co-exist on the same board.
* BOARD may be partially populated as 56K.

256K S-100 SOLID STATE DISK SIMULATOR!

WE CALL THIS BOARD THE "LIGHT-SPEED-100" BECAUSE IT OFFERS AN ASTOUNDING INCREASE IN YOUR COMPUTER'S PERFORMANCE WHEN COMPARED TO A MECHANICAL FLOPPY DISK DRIVE.

PRICE CUT!



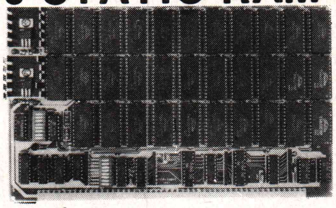
FEATURES:
* 256K on board, using + 5V 64K DRAMS.
* Uses new Intel 8203-1 LSI Memory Controller.
* Requires only 4 Dip Switch Selectable I/O Ports.
* Runs on 8080 or Z80 S100 machines.
* Up to 8 LS-100 boards can be run together for 2 Meg. of On Line Solid State Disk Storage.
* Provisions for Battery back-up.
* Software to mate the LS-100 to your CP/M* 2.2 DOS is supplied.
* The LS-100 provides an increase in speed of up to 7 to 10 times on Disk Intensive Software.
* Compare our price! You could pay up to 3 times as much for similar boards.

BLANK PCB (WITH CP/M* 2.2 PATCHES AND INSTALL PROGRAM ON DISKETTE)
$69.95
(8203-1 INTEL $29.95)

$259.00
#LS-100 (FULL 256K KIT)

64K SS-50 STATIC RAM

$149.95 (48K KIT)



NEW!

LOW POWER!

RAM OR EPROM!

BLANK PC BOARD WITH DOCUMENTATION
$52

SUPPORT ICs + CAPS
$18.00

FULL SOCKET SET
$15.00

56K Kit $169

64K Kit $199

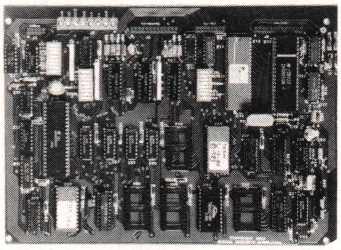
ASSEMBLED AND TESTED ADD $50

FEATURES:
* Uses new 2K x 8 (TMM 2016 or HM 6116) RAMs.
* Fully supports Extended Addressing.
* 64K draws only approximately 500 MA.
* 200 NS RAMs are standard. (TOSHIBA makes TMM 2016s as fast as 100 NS. FOR YOUR HIGH SPEED APPLICATIONS.)
* Board is configured as 3-16K blocks and 8-2K blocks (within any 64K block) for maximum flexibility.
* 2716 EPROMs may be installed anywhere on Board.
* Top 16K may be disabled in 2K blocks to avoid any I/O conflicts.
* One Board supports both RAM and EPROM.
* RAM supports 2MHZ operation at no extra charge!
* Board may be partially populated in 16K increments.

THE NEW ZRT-80

CRT TERMINAL BOARD!

A LOW COST Z-80 BASED SINGLE BOARD THAT ONLY NEEDS AN ASCII KEYBOARD, POWER SUPPLY, AND VIDEO MONITOR TO MAKE A COMPLETE CRT TERMINAL. USE AS A COMPUTER CONSOLE, OR WITH A MODEM FOR USE WITH ANY OF THE PHONE-LINE COMPUTER SERVICES.



FEATURES:
* Uses a Z80A and 6845 CRT Controller for powerful video capabilities.
* RS232 at 16 BAUD Rates from 75 to 19,200.
* 24 x 80 standard format (60 Hz).
* Optional formats from 24 x 80 (50 Hz) to 64 lines x 96 characters (60 Hz).
* Higher density formats require up to 3 additional 2K x 8 6116 RAMS.
* Uses N.S. INS 8250 BAUD Rate Gen. and USART combo IC.
* 3 Terminal Emulation Modes which are Dip Switch selectable. These include the LSI-ADM3A, the Heath H-19, and the Beehive.
* Composite or Split Video.
* Any polarity of video or sync.
* Inverse Video Capability.
* Small Size: 6.5 x 9 inches.
* Upper & lower case with descenders.
* 7 x 9 Character Matrix.
* Requires Par. ASCII keyboard.

BLANK PCB WITH 2716 CHAR. ROM, 2732 MON. ROM
$49.95

SOURCE DISKETTE - ADD $10

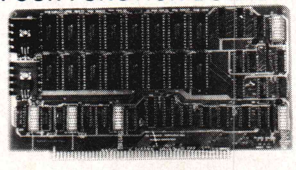
SET OF 2 CRYSTALS - ADD $7.50

WITH 8 IN. SOURCE DISK! (CP/M COMPATIBLE)

$99.95
ZRT-80 (COMPLETE KIT, 2K VIDEO RAM)

32K S100 EPROM/STATIC RAM

NEW! FOUR FUNCTION BOARD! **NEW!**



EPROM II
FULL EPROM KIT
$69.95

A&T EPROM
ADD $35.00

BLANK PC BOARD WITH DATA
$39.95

SUPPORT IC'S PLUS CAPS
$16

FULL SOCKET SET
$15

We took our very popular 32K S100 EPROM Card and added additional logic to create a more versatile EPROM/RAM Board.

PRICES SLASHED!

FEATURES:
* This one board can be used in any one of four ways:
  A. As a 32K 2716 EPROM Board
  B. As a 32K 2732 EPROM Board (Using Every Other Socket)
  C. As a mixed 32K 2716 EPROM/2K x 8 RAM Board
  D. As a 32K Static RAM Board
* Uses New 2K x 8 (TMM2016 or HM6116) RAM's
* Fully Supports IEEE 696 Buss Standard (As Proposed)
* Supports 24 Bit Extended Adressing
* 200 NS (FAST!) RAM'S are standard on the RAM Kit
* Supports both Cromemco and North Star Bank Select
* Supports Phantom
* On Board wait State Generator
* Every 2K Block may be disabled
* Addressed as two separate 16K Blocks on any 64K Boundary
* Perfect for MP/M* Systems
* RAM Kit is very low power (300 MA typical)

32K STATIC RAM KIT — $109.95
For RAM Kit A&T — Add $400

Digital Research Computers

P.O. BOX 461565 • GARLAND, TEXAS 75046 • (214) 225-2309

TERMS: Add $3.00 postage. We pay balance. Orders under $15 add 75¢ handling. No. C.O.D. We accept Visa and MasterCharge. Tex. Res. add 5-1/8% Tax. Foreign orders (except Canada) add 20% P & H. Orders over $50, add 85¢ for for insurance.

ALL SALES ARE MADE SUBJECT TO THE TERMS OF OUR 90 DAY LIMITED WARRANTY. A COPY OF THIS WARRANTY IS AVAILABLE FREE, ON REQUEST.

*TM OF DIGITAL RESEARCH INC. (CALIF.)

WE ARE NOT ASSOCIATED WITH DIGITAL RESEARCH INC. (CALIF.) THE SUPPLIERS OF CPM SOFTWARE

Circle **no. 33** on reader service card.

```
10 INPUT "Program to send?";F$
20 OPEN "I",1,F$
30 INPUT "Now exit Transparent
   Mode and do a LOAD.";Z$
40 WHILE NOT EOF(1):LINE IN-
   PUT #1,A$:PRINT A$:WEND
```

Admittedly, this way of LOADing and SAVEing is a fairly complex procedure, but it allows you to save your programs on disk while keeping the interpreter itself small. If your ECB isn't connected to another computer, you probably could change the AUXIN and AUXOUT subroutines to use the cassette interface. (I haven't tried it, though, so *caveat emptor!*)

For other 68000 systems, you will have to modify only the OUTC, INC, AUXOUT, AUXIN, and BYEBYE routines at the end of the interpreter program. In addition, you must put the address of the first unavailable memory location above BASIC into the location ENDMEM. BASIC programs are SAVEd in a form that can be stored as ASCII text and read back quickly by the 68000; if your storage device can't handle the present format or if you would like the program saved in a human-readable form, you need modify only the SAVE and LOAD subroutines.

One warning: I wrote the DIRECT and EXEC routines assuming that the interpreter itself would be somewhere in the first 64K of memory ($0 to $FFFF). If you move it above 64K, you will have to modify the EXEC routine and check the rest of the code carefully to make sure the addressing modes are correct.

A.=ABS
C.=CALL
F.=FOR
GOS.=GOSUB
G.=GOTO
IF=IF
I.=INPUT
L.=LIST
LO.=LOAD
N.=NEW
N.=NEXT
P.=PEEK
PO.=POKE
P.=PRINT
REM=REMARK
R.=RETURN
R.=RND
R.=RUN
S.=SAVE
S.=SIZE
S.=STEP
S.=STOP
TO=TO
no keyword = LET

Table

Evaluation

I am quite pleased with how the interpreter turned out. Even though I added extra error checking, lower-case conversion, and more commands and extended the variable size to 32 bits, the whole thing still fits inside 3K of memory. I ran the Sieve of Eratosthenes benchmark program on this interpreter and on the Sherry Brother's CP/M tiny BASIC with the following results:

68000 at 4 MHz	Z80 at 4 MHz
2670 seconds	3000 seconds

Although I adjusted the results for the usual 10 iterations of the basic algorithm, I actually ran the program only for one iteration to keep running times within a practical limit. This tiny BASIC may not be a speed demon, but it does beat Applesoft and PET BASIC at running the Sieve benchmark. I should add that I compressed the Sieve program listing to the maximum for speed considerations; I normally use more spaces and some comments so that I can figure out later what the program was supposed to do!

Of course, many improvements can be made given more available memory. My Educational Computer Board has 32K of memory, so I probably will add such things as more variables, strings, and keyword tokenization. The last is a method used by most BASIC interpreters to compress keywords such as LET and PRINT into single bytes. This would greatly speed up the interpreter while using less memory to store the BASIC program.

Availability

By the time you read this, the interpreter source code and some example programs should be available on a couple of the RCP/M bulletin board systems in my area:

Meadowlark RCP/M (403) 484-5981

Edmonton RCP/M (403) 454-6093

The Edmonton RCP/M accepts both 300 and 1200 baud, but the Meadowlark system allows access to its CP/M area only at 1200 baud. Both systems run 24 hours a day.

The interpreter source code is known as TBI68K.AQM, which is a "squeezed" text file. If you don't have a MODEM7 type program and a way to unsqueeze this file, you can use these systems' LIST command to list out the source code while you capture it with a telecommunications program.

A short documentation file, TBI68K.DQC, and some sample programs, TBIPROGS.LBR, are also available. The latter is a CP/M library file, which contains several programs. You can list the library's contents with the LDIR command and extract individual programs using either the systems' XMODEM or LTYPE commands. The Quelo cross-assembler is also sometimes available on these systems under the names A68K.COM and A68K.DOC.

Although I'd prefer that you obtain the source code from one of the above sources, for $20 I can provide the code in the following forms: 8-inch CP/M SSSD diskette, 5-inch Osborne or Apple CP/M diskettes, or a paper listing.

If you find any bugs in the interpreter or have any questions, please write to me or contact me on the above RCP/M system.

DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 194.**

LISP FOR THE IBM PERSONAL COMPUTER.

THE PREMIER LANGUAGE OF ARTIFICIAL INTELLIGENCE FOR YOUR IBM PC.

- **DATA TYPES**
  Lists and Symbols
  Unlimited Precision Integers
  Floating Point Numbers
  Character Strings
  Multidimensional Arrays
  Files
  Machine Language Code
- **MEMORY MANAGEMENT**
  Full Memory Space Supported
  Dynamic Allocation
  Compacting Garbage Collector
- **FUNCTION TYPES**
  EXPR/FEXPR/MACRO
  Machine Language Primitives
  Over 190 Primitive Functions
- **IO SUPPORT**
  Multiple Display Windows
  Cursor Control
  All Function Keys Supported
  Read and Splice Macros
  Disk Files
- **POWERFUL ERROR RECOVERY**
- **8087 SUPPORT**
- **COLOR GRAPHICS**
- **LISP LIBRARY**
  Structured Programming Macros
  Editor and Formatter
  Package Support
  Debugging Functions
  .OBJ File Loader
- **RUNS UNDER PC-DOS 1.1 or 2.0**

IQLISP

5¼″ Diskette and Manual	$175.00
Manual Only	$ 30.00

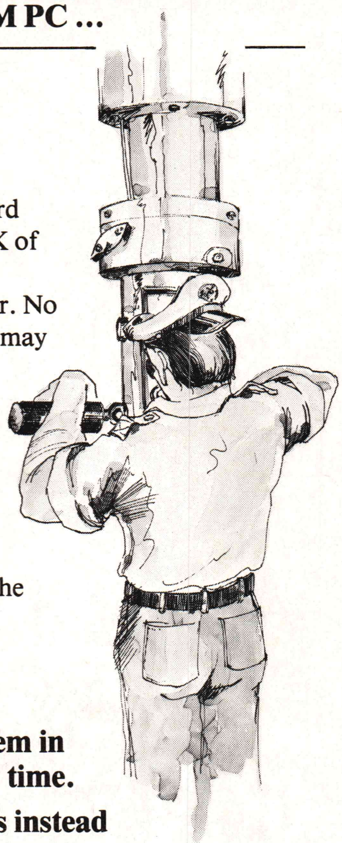
Integral Quality

P.O. Box 31970
Seattle, Washington 98103-0070
(206) 527-2918

Washington State residents add sales tax.
VISA and MASTERCARD accepted.
Shipping included for prepaid orders.

Debugging Bugging You?

Torpedo program crashes and debugging delays with debugging dynamite for the IBM PC …



UP PERISCOPE!

First, you install the hardware.

The hardware's a special memory board that fits in a PC expansion slot. Its 16K of write-protected memory contains Periscope's resident symbolic debugger. No runaway program, however berserk it may be, can touch this memory!

Then you UP PERISCOPE.

Use Periscope's push-button break-out switch to interrupt a running program … even when the system's hung! Periscope supports Assembly, BASIC, C and Pascal. In addition to the usual debugging capabilities, some of Periscope's features are:

Stop your system in its tracks at any time.

Use symbol names instead of addresses.

Run a program on one monitor and debug on another.

Monitor your program's execution with Periscope's comprehensive breakpoints.

Debug memory-resident programs.

Put your time to better use.

The Periscope system is $295. It carries a 30-day money-back guarantee and includes the memory board, remote break-out switch, debugger software, 100-page manual, and quick-reference card. The memory board is warranted for one year. A demonstration disk is $5.00.

System requirements for Periscope are an IBM PC, XT or Compaq, PC-DOS, 64K RAM, 1 disk drive and an 80-column monitor. For MasterCard and Visa orders only, call 800/421-5300 (ext. R96) 24 hours a day. For additional information, call 404/256-3860 from 9 AM to 5 PM Eastern Time.

Get your programs up and running;

UP PERISCOPE!

Data Base Decisions / 14 Bonnie Lane /Atlanta, GA 30328

Circle **no. 30** on reader service card.

An Enhanced ADFGVX Cipher System

by C. E. Burton

Cryptography is too important to be left to the NSA.

In another article, I described a public key system based on the RSA algorithm.[1] The major disadvantage of that algorithm was that it was not very fast, either in key generation or in encryption/decryption. However, if you need the security, you probably should use the RSA cryptography system or the DES cryptosystem.

Most information, however, has a time value associated with it. Thus, it is not necessary to use a RSA or a DES cryptography system on all messages. For example, if a military operation were going to be mounted within a month, we could encrypt messages relating to the operation using any cryptographic system that we can show to be unbreakable over that period of time (plus some "slop" for weather, delays, etc.).

Private key systems are fast and can be difficult to break. The cryptography system described in this article is based on a method developed by the Germans during World War I. David Kahn, in his book *The Code Breakers* (MacMillan, 1967), indicated that this field cipher system was probably the toughest then developed.[2] The original cipher was termed the ADFGX system, and it took Painvin, a French cryptoanalyst, over a month to decode the first batch of messages that the Germans encrypted.

The ADFGX system, however, has the same problem that the DES system or any other private key system has: the key must be known to both the encryptor and the decryptor. Muller-Schloer has suggested one way around this problem.[3] Having some way to send the key with the message would eliminate the logistics of key distribution. The question becomes: How can we do that without giving everything away? The answer is really simple! We encrypt the message with a private key system. Next we encrypt the private key, using a public key system, and append the encrypted form of the private key to the message.

We now have the best of both worlds. The private key system can encrypt/decrypt a message rapidly (in hardware or in software), and we can even change the key for every message. Because the private key is short, it does not take long to encrypt/decrypt it using a public key system.

This article describes an enhanced version of the ADFGX cipher system: the cipher, the character substitution method, the use of a key or password to do an irregular columnar transposition, a run-length data compression method, and some of the housekeeping functions performed in the encryption/decryption process. The substitution method contains a "card"-shuffling algorithm and a random number generator. The language used to write these routines is C.

Before we get into the details of my implementation, we should look at the original German ADFGX cipher system to get a feel for where we will be heading.

The ADFGX Cipher

The ADFGX cipher was named after the letters of the Morse code used in the transmission of the encrypted messages. Be-

---

Charles E. Burton, 13284 W. Utah Circle, Denver, CO 80228.

cause these particular characters are quite distinctive, the Germans felt that, even if noise partially garbled the message, these characters could not be confused. For those of you unfamiliar with Morse code, it represents the alphabet by a series of dots and dashes. The telegraph system sent and received messages translated into Morse code, and after Marconi developed the radio, Morse code was used over the air waves. The Morse code representation of the cipher letters is:

A .–
D –..
F ..–.
G ––.
X –..–

With these letters, the Germans could represent a scrambled alphabet in a 5 × 5 square matrix (leaving out one letter, say, Q). The substitution matrix might have looked like this:

	A	D	F	G	X
A	O	D	C	M	S
D	U	Y	X	L	H
F	T	A	N	W	Z
G	E	J	G	I	P
X	F	V	B	R	K

The next item required was a key, which is used to do a columnar transposition. We assign each letter in the key a column number. Then we sort the key alphabetically by character to get the column order for transmission. Let us choose a key: DOG MEAT. Squeezing out the blanks and alphabetically sorting the letters of the key, we get ADEGMOT. The sorted column order is 6/1/5/3/4/2/7:

D	O	G	M	E	A	T
1	2	3	4	5	6	7

becomes

A	D	E	G	M	O	T
6	1	5	3	4	2	7

Now, we need a message to see how all of this works. Let's use the message, "Meet me at the zoo at ten." After removing all of the spaces and punctuation, we have:

MEETMEATTHEZOOATTEN

Next we take each letter, in turn, and pass it through the ADFGX substitution table to get the row and column address (in the form of a row/column pair using the letters ADFGX). For example, the letter M would become AG (row A, column G):

M	E	E	T	M	E	A	T	T	H	E
AG	GA	GA	FA	AG	GA	FD	FA	FA	DX	GA

Z	O	O	A	T	T	E	N
FX	AA	AA	FD	FA	FA	GA	FF

We place these letter pairs in a table that has columns headed by the key letters (DOG MEAT). The key transposition table is loaded a row at a time:

D/1	O/2	G/3	M/4	E/5	A/6	T/7
A	G	G	A	G	A	F
A	A	G	G	A	F	D
F	A	F	A	D	X	G
A	F	X	A	A	A	A
F	D	F	A	F	A	G
A	F	F				

Message			
M	E	E	T
	M	E	A
T	T	H	E
	Z	O	O
A	T	T	E
	N		

Finally, we pull the columns out and rearrange them as rows in the previously determined order (6/1/5/3/4/2/7):

6	1	5	3
AFXAA	/AAFAFA	/GADAF	/GGFXFF /

4	2	7
GAAA	/GAAFDF	/FDGAG

The message is now encrypted and ready for transmission (without the "/"s). Note that, although the message length has doubled, sending the encrypted message in this fashion gave the Germans benefits that outweighed this disadvantage: the distinctive nature of the letters transmitted. Certainly they could have run the transposed message back through the substitution table to get a message of length equal to the original:

AF	XA	AA	AF	AF	AG	AD	AF	GG	FX
C	F	O	C	C	M	D	C	I	Z

FF	AG	AA	AG	AA	FD	FF	DG	AG
N	M	O	M	O	A	N	L	M

But this would mean using the whole alphabet rather than five letters and increase the chance of error introduction at two additional stages (one at the sending end and one at the receiving end). Remember, there were no computers during World War I. On the other hand, had the Germans taken this extra step, in all likelihood it would have taken much longer to break the cipher.

The decryption process follows the above steps in the reverse order: the received letters are put into the key table in their original columns; character pairs are read out from the rows; and these pairs are used in the substitution table to get the original message. Note that the receiver must know two pieces of information to decipher the message: the key and the substitution table.

The letter V was added to the original ADFGX formulation to allow all 26 letters and 10 digits to be used; the substitution table produced by the ADFGVX cryptosystem was a 6 × 6 matrix. Again the 36 characters were put into the substitution matrix in a random order. The same procedure to encrypt/decrypt was used.

Enhanced ADFGVX

I have enhanced the basic ADFGVX system in several areas. First, because it is useful to be able to encrypt/decrypt text, as well as data, programs or anything else, we must be able to deal with byte-wide data. A substitution table that can handle characters from 0 to 255 (00 to FF hex) must be a 16 × 16 matrix.

Second, because the "messages" will be longer than a few "words," one columnar transposition will not do; we must block the "message" into something more suitable. Each block will be passed through the algorithm for encryption and decryption. Because each letter of the message will be expanded to two letters by the substitution step, a transposition table with an even number of characters is desirable (but not mandatory). To make the transposition more difficult to figure out, the table must not end on an even square, such as 25 or 36. That way the messages are not blocked into a form that provides information about the ciphering method. Foster has shown that the use of incomplete or irregular columnar transposition makes the deciphering task much harder.[4]

Third, some simple data compression, especially for text, would reduce storage capacity and transmission time. We also will do the backward substitution operation at the end so that the message size does not double—we have a computer that should not make the mistakes that a human might.

A couple of additions could make this implementation even more difficult to attack and break. First, make the incomplete columnar transposition variable. For example, the previous key transposition table had seven columns. Say that we need between 4 and 10 rows in the transposition table before transposing. We could fill the transposition table with an even number of characters from the substitution operation so that 28 (7 × 4) to 70 (7 × 10) of these characters are loaded into the transposition table before each column transposition operation. For instance, choose the repeating character count sequence: 46, 62, 30, 54, and 38. On the first pass, we load the first 23 (46 / 2) row/column pairs into the transposition matrix; on the next pass, we load the next 31 row/column pairs; and so on, cycling through the five character count sequences. The main problem with this variability option is that it involves another piece of information that both the sending and receiving parties must know.

The second addition, which involves a double irregular columnar transposition, again requires that another piece of information be available to both parties. To apply this option, we do everything as usual, using one key to do the first transposition, then we use a second key to form a second transposition table (just as we did with the first key); we fill the second transposition table with the output of the first transposition table. Next we perform the second transposition, and finally we do the backward substitution to get the encrypted message. The technique of double transposition has been used as a military cipher system ("U.S. Army Transposition") because of the difficulty in deciphering the result. Foster indicated that he knew of no method to crack the double transposition technique by hand or by microcomputer.

Figures 1 and 2 (page 51) present the structure charts for the encryption and decryption processes, respectively. Rather than have each of the boxes represent a separate routine, I have combined many of these routines together. Those boxes with an asterisk (*) above them indicate separate modules; their names are given below the box. Those boxes without the special notation appear in their parent box. The looping arrow at the bottom of a box indicates repetition, and the diamond indicates a decision (the child may or may not be called, depending on the outcome of the decision).

We now are ready to discuss the various procedures. Even though I designed the software in a top-down fashion, I will introduce the modules in a somewhat bottom-up fashion.

Substitution Matrix

We must generate a 16 × 16 matrix (256-entry table) to get our character substitutions. Rather than use characters (e.g., ADFGVX or equivalent) to represent the rows or columns, we will use the row and column numbers (0 to 15). We must fill the matrix positions in a random order. A simple card-shuffling routine can be used to do this. We also must overcome one minor problem: how can we find the "card" value, given its position in the "deck," and how can we find the position in the "deck" of a "card" of a given value. We need these answers to perform both the forward and backward substitution procedures.

The routines that handle the matrix filling are shuffle( ) and ran( ). shuffle( ) takes a linear array (having 256 character positions) and fills the entries in linear order. Then it passes through the "deck" and swaps the current "card" with another card in a random position as determined by a call to ran( ).

Once the "deck" has been shuffled, it is scanned to find the position of each "card" within it. The result is two linear arrays:

position[I] – holds the position of card I within the deck
value[J] – holds the value of the Jth card in the deck

Given position[ ], we can find the row and column position of a character (or perform the forward substitution) by:

```
row = position[char] / 16
column = position[char] % 16
   (i.e., position[char] mod 16)
```

Given value[ ], we can find the character occupied by a row and column (or perform the backward substitution) by:

```
char = value[ 16 * row + column]
```

Note that we are using a linear array to represent a matrix.

ran(MAX) returns a random integer between 0 and (MAX−1). We can alter the random number generation sequence by changing the initial value of IY to be greater than 0 and less than 2796203. One possibility might be to declare IY as a global variable and to ask for a seed value in the main program. Remember, however, both the sender and the receiver *must* have the same starting seed to produce identical substitution tables. For the same reason, the substitution table must be generated before any other use is made of ran( ).

Irregular Columnar Transposition

From the ADFGX example, we found that the transposition involved several steps. First, given a password or key, we must sort the password on a character-by-character basis. This job is performed by pwsort( ). In pwsort( ), I have limited the length of the password to values greater than 4 and less than 11. If the password is less than 5, the routine gives an error message and exits to the operating system. If the password is greater than 10, the password is truncated to the first 10 characters. You can alter the routine to use other password lengths. Once the password is validated, we sort

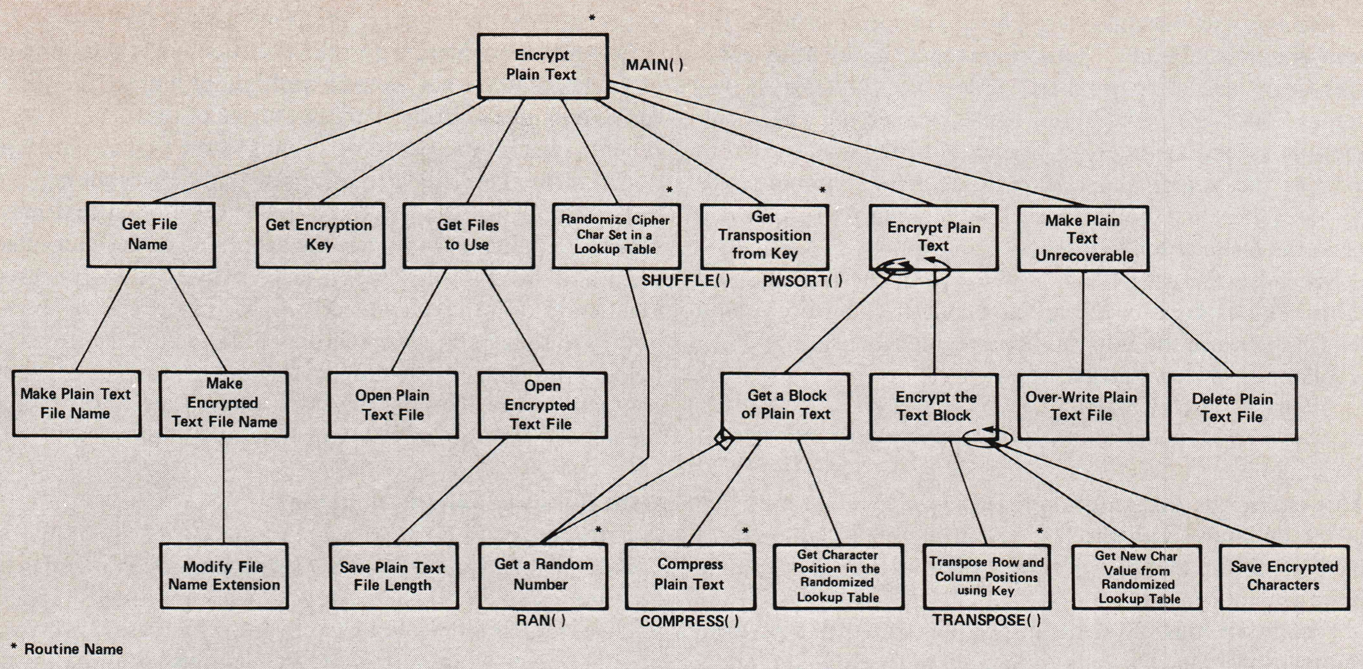


Figure 1

Encrypt Structure Chart

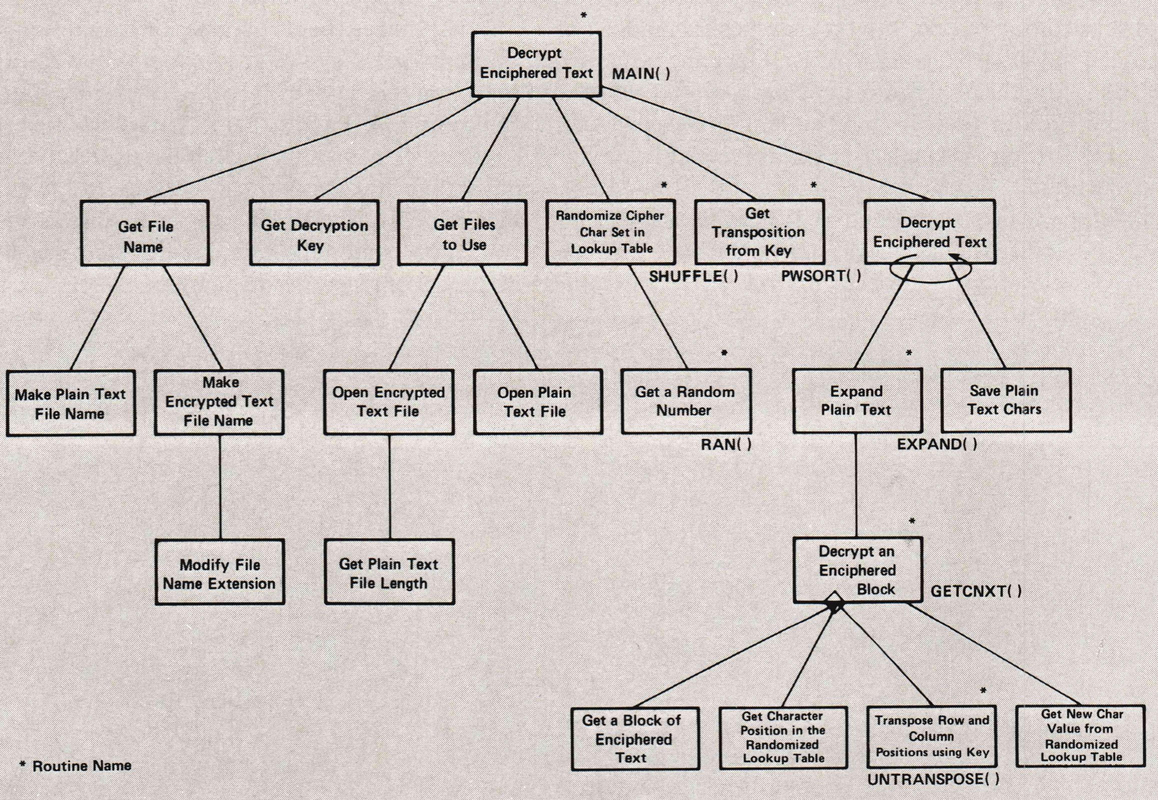


Figure 2

Decrypt Structure Chart

the characters individually using a simple bubble sort.

The array pwcolumnorder[ ] holds the sorted order of the transposition columns. This sort arranges the columns in descending (largest to smallest) order; i.e., DOGMEAT becomes TOMGEDA. Variations could arrange the sorted columns in ascending order, inside-out, outside-in, and so on, as long as the sender and receiver agree. pwcolumnorder[0] provides the first column and pwcolumnorder[pwlen−1] gives the last column to extract.

Second, we must determine the size of the transposition table. This is done in the main( ) routine. I have picked a length to ensure the columns are irregular by using:

blklen = pwlen * pwlen + pwlen / 2

blklen + = odd(blklen) ? 1 : 0

(i.e., if blklen is odd then increment blklen else leave it alone)

This makes the transposition matrix (pwlen + 1) rows and pwlen columns, with the last row incomplete. Note that I have made the number of entries even so that row/column pairs are not broken.

Finally, we must fill the transposition matrix and perform the transposition. For encryption, we must enter the information from the forward substitution matrix into the transposition matrix by rows and in the substitution row/column order. Once the transposition matrix is filled, the routine transpose( ) is called to pull out the columns in sorted order. These new row/column pairs are passed through the backward substitution and then saved as the encrypted message.

For decryption, we must enter the information from the forward substitution matrix into the transposition matrix by columns and in the sorted column order. Once the transposition matrix is full, the dual routine untranspose( ) is called to pull out the rows in their original order. These new row/column pairs are run through a backward substitution and saved as the plain text message.

Actually, the transposition matrix is filled in a linear order, and the transpose( ) and the untranspose( ) routines take care of obtaining the proper ordering of the rows and columns. The transposition matrix-filling operation produces the message blocking that I mentioned previously.

One minor problem with this operation is the handling of messages that are not an even multiple of blklen (the usual case). A simple solution is to pad the end of the message with random characters. However, to make sure that the decryption returns the original message without the padding, we must arrange for the length of the original plain text message to be found and saved at the beginning of the encrypted file. I have done this by using a union called filelen. It is used identically to an "equivalence" in Fortran. Because the C library routine putc( ) handles only bytes, I built the union to hold a long integer (the file length) and an equivalent-sized character array. This construction allowed me to load/save the file length from/to a file in a byte-at-a-time mode.

Data Compression/Expansion

A useful program is available on most CP/M-based bulletin board systems. Called SQ/USQ (squeezer), it allows files to be compressed and expanded so that you can save disk space. Richard Greenlaw wrote the software, which uses two types of compression: run length compression and Huffman coding. The achievable compression ranges from 3% to 70%, depending on the makeup of the file. I have borrowed and modified the run length compression routine to perform my data compression. The expansion routine that I used is modeled after the compression routine. (Although Greenlaw's USQ program has a more efficient expansion routine, I chose to do my expansion differently.)

What makes these compression/expansion routines interesting is that they are good examples of state machine routines. Figures 3a and 3b (below) show the state diagrams for the compress( ) and expand( ) routines, respectively. These routines show how the switch-case construct can be used to advantage in representing the states. In the figures, the bubbles represent the states and the arrows represent the state transition conditions. Similar directed graphs are used in electrical engineering to represent logic circuit operations.

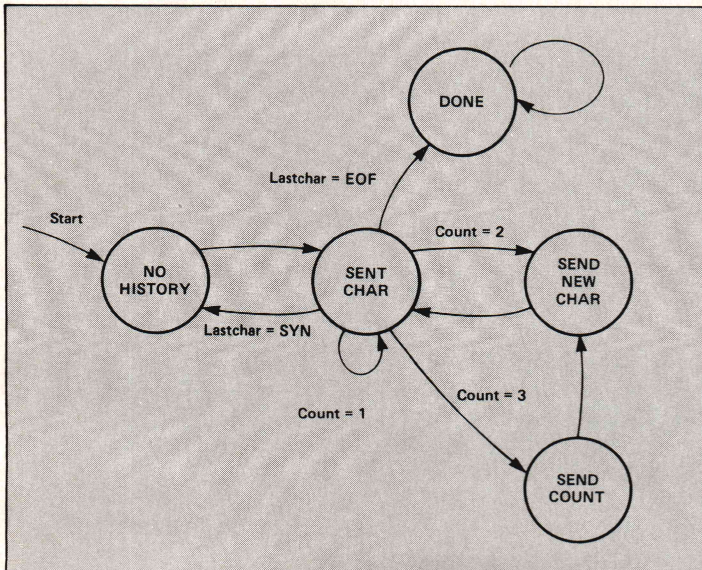


Figure 3a
COMPRESS( ) State Diagram

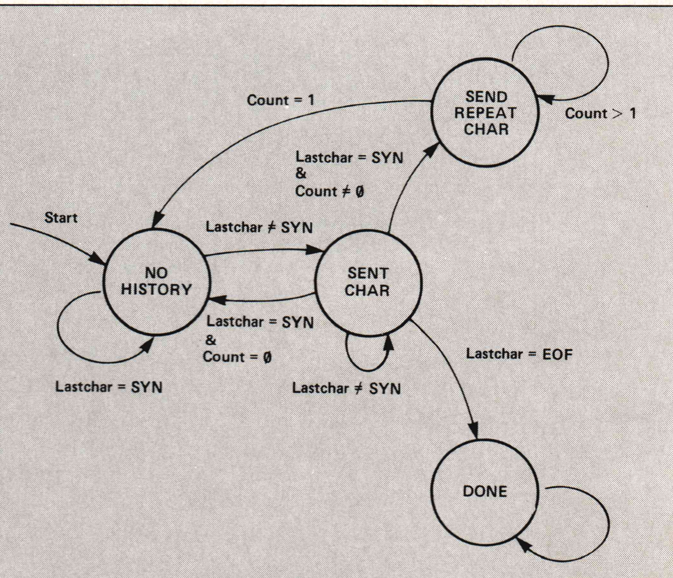


Figure 3b
EXPAND( ) State Diagram

Software dealing with control mechanisms also uses state diagrams. Comparing the state diagrams with the routines provides an easy understanding of the code.

Basically, the run length compression routine looks at the character stream. If three or more adjacent characters are the same, the routine counts the like characters and substitutes a three-byte string that incudes the character, a special character indicating compression has occurred, and the character repeat count; i.e., <char> SYN <count>. To handle the circumstance where the special character occurs in the character stream, the routine generates the special character and a character count of zero; i.e., SYN <0>. Finally, if an end of file (EOF) occurs, the routine responds with an EOF thereafter. The expansion operation essentially reverses the process.

Housekeeping Operations

One of the nice features of C is that it allows you to read information from the command line that calls the program into execution. I have used this capability in both the encryption and the decryption programs. With either program, you specify the execution program name, followed by the plain text filename, followed by the password or key. Most of the early housekeeping involves picking up the filename and password.

Once the plain text filename is retrieved, the encrypted filename is generated from the plain text filename by modifying the filename extension. If the plain text filename extension has at least one character, a Y is substituted for the second letter of the extension. If there is no extension, an extension of .YYY is appended to the filename. For example, if the plain text filename is MESSAGE.TXT, then the encrypted filename becomes MESSAGE.TYT.

The encryption program checks for the existence of the plain text source file, and the decryption program checks for the existence of the encrypted source file. If the source file cannot be found and opened for reading, the programs exit to the operating system after generating an error message. Similarly, if the target file is opened for writing and a problem occurs, a message is generated and an exit takes place. After taking care of the files, the password is retrieved and is checked using pwsort( ), as already described.

As indicated previously, the length of the plain text file must be found. The encryption process determines the plain text file length using the fseek( ) and ftell( ) functions found in the C library. It saves the length at the head of the encrypted file. The decryption process reads the file length from the encrypted file and uses the value to determine when the end of the plain text file has occurred, exclusive of the padding characters.

Once these steps are complete, the encryption/decryption process proceeds as already explained. The arrays BUF1[ ] and BUF2[ ] hold the row/column pairs that result from the substitution process. These arrays contain a block of message, the length of which is twice the length of the plain text or encrypted text block. BUF1[ ] holds the forward substitution pairs; BUF2[ ] contains the result of the transposition that is ready for submission to the backward substitution. The encryption program uses these arrays in the main( ) routine, and the decryption program uses them in the getcnxt( ) routine. Note that these arrays are large enough for the largest password used in the columnar transposition (plus a little bit).

After the encryption/decryption process is complete, you no longer need to keep the source file, so it is erased. When you are debugging these programs, you will probably want to inhibit the unlink( ) function that performs this file deletion.

One final operation is performed in the encryption program prior to source file deletion. Because certain programs can "unerase" or look at a disk on a sector-by-sector basis, and we do not want to leave any traces of the original plain text file, we must overwrite the entire plain text file with something. I have chosen to use the character F6 hex, but any byte character will do. Again you will probably want to inhibit this operation while debugging. (There is no need to overwrite the encrypted file, since it is "gibberish" anyway.)

Example

Figure 4a (page 55) is the original plain text file that I encrypted and subsequently decrypted. I have shown the encrypted text file (Figure 4b, page 55) and the decrypted text file (Figure 4c, page 56) in both hex and ASCII representations; this is essentially the same representation used by the D command under DDT. Note that the encrypted file's first four bytes provide the length of the original plain text file. It is after that point (from the fifth byte to the end of the file) that the encrypted text of the original plain text file occurs. The plain text file has several repeated character sequences to demonstrate data compression.

To duplicate my results, let's call the plain text file ADFGVX.TXT and use the password GERMANS. To encrypt ADFGVX.TXT, you type in the command line:

ENCRYPT adfgvx.txt germans

This produces the encrypted file ADFGVX.TYT and destroys ADFGVX.TXT. To decrypt the encrypted file, you type the command line:

DECRYPT adfgvx.txt germans

This produces the decrypted file ADFGVX.TXT and erases ADFGVX.TYT. You enter the command lines at the operating system prompt; e.g., at the >A under CP/M. You must compile, assemble, and link the programs so that the encryption program has a name of ENCRYPT.COM and the decryption program has the name DECRYPT.COM.

Note that this implementation does not remove spaces, punctuation, and so on. Because the substitution table covers the entire byte-wide character set, what you put in will come out at the other end unaltered. You do not have to squeeze together then break up the text, as we had to do with the original German cipher. In addition, you can encrypt/decrypt any file.

Conclusion

Although this cipher is not as secure as a RSA or a DES cryptosystem, its character set scrambling makes it indistinguishable from these other methods. According to Kahn, this cipher is not easily broken. The advantages are that the cipher is simple to implement and fast in execution. The major disadvantage is that both the sender and the receiver must know the key or password and the substitution table ordering. As I pointed out, you can overcome this difficulty by using a public key system to encrypt the key and table

First of all, let's generate a string of repeated characters:

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
```

These sequences will be converted by the data compression routine to the following three character representation:

```
<x> SYN <52> = 78 16 34   (in hex)
<Y> SYN <52> = 59 16 34   (in hex)
```

Notice that we are not restricted to upper case letters and digits only. This implementation does not have to remove spaces and punctuation. Thus, all ASCII characters (0 to 127) can be used, as well as the byte values from 128 to 255.

The encrypted file will not have a one-to-one correspondence between these characters and those produced by the encryption process, i.e. letter frequency analysis will not help. This result is due to the transposition, which may replace an "a" in one case by a "Y" and in another case by a "*", etc.

Figure 4a

Plain Text to be encrypted. File Name is ADFGVX.TXT.

```
80 04 00 00 56 FA B0 CD  B7 AC 8D EB 32 6C 67 59   ....V.......21gY
F7 9E 20 9E 4B A0 8D 26  20 25 B0 75 5C 83 6F 60   .. .K..& %.u\.o`
09 AC 28 46 E1 32 26 B9  FE 0B 82 5C 5C 6D 0B 20   ..(F.2&....\\m.
9E 20 0B 2C 7C 65 92 9E  3E F9 16 F6 0B 3D 54 54   . .,|e..>....=TT
B4 79 D7 E8 9D E6 C2 2B  77 27 20 41 5E 2C AE 7A   .y.....+w' A^,.z
C2 F2 0A 65 F7 A7 C2 31  31 B3 D2 20 54 F2 AC 8D   ...e...11.. T...
A8 53 89 F7 F3 0D 54 C6  CD 33 31 0C 31 25 8D 8D   .S....T..31.1%..
37 FB 53 31 CF 48 20 00  65 18 A0 31 57 83 DF 0B   7.S1.H .e..1W...
81 20 AA 20 7A FB AB BD  0B FA 5C A0 55 62 3E 68   . . z.....\.Ub>h
20 65 90 82 C2 FB 5C 56  44 0B 35 C4 20 52 89 31    e....\VD.5. R.1
9E 65 6E 65 46 4A 00 35  57 57 EB ED 5F CF FF 6D   .eneFJ.5WW.._..m
20 F7 F7 0B 8D 5D BD 87  87 B1 25 B4 91 42 AC 5F    ....]....%..B._
BA 35 E2 2C 86 15 0B 0B  FF BD FB 62 E4 57 8D 2C   .5.,.......b.W.,
8D E8 8D 9E 5E 6B CA 04  C2 34 F7 25 D2 42 6D 55   ....^k...4.%.BmU
AC 91 E8 3D 62 38 08 0D  31 57 00 33 0E 79 92 8D   ...=b8..1W.3.y..
29 C1 6D B7 E6 FB C2 AC  90 FB 00 53 67 90 12 CD   ).m........Sg...
0E 04 58 D2 0B 20 FF 4F  CA 36 20 CF F7 25 04 AC   ..X.. .O.6 ..%..
6D 39 BB 91 86 90 FB 38  79 20 31 57 20 C2 09 79   m9.....8y 1W ..y
18 BD 81 40 25 31 C2 52  65 A8 D2 9E AA 53 65 C6   ...@%1.Re....Se.
5F 90 71 F7 77 0D 8E 57  AC 33 5C 6D 57 F7 40 05   _.q.w..W.3\mW.@.
20 84 FB 71 BD 41 9E ED  65 FB 20 9E BD 9E 62 40    ..q.A..e. ...b@
7F AB 1B FF 35 F7 49 BD  0B AE ED 6D 3C 62 65 AA   ~...5.I....m<be.
20 3E 20 CD C2 FB 38 E4  44 0B 9D DD 20 31 8D FB    > ...8.D... 1..
0B 6C AA 48 35 C7 93 23  25 25 09 C9 CB 63 CD B7   .l.H5..#%%...c..
CD 8B A0 8D 70 69 09 82  0E 3C 23 A5 6F 20 ED ED   ....pi...<#.o ..
20 DF 0B 32 0B 9E 31 FF  74 35 FB 3F 20 61 8D C7    ..2..1.t5.? a..
0C 6D 9E 31 E2 D2 58 6C  8D 43 FB FB 8D 16 32 40   .m.1..Xl.C....2@
8D 31 65 61 C2 5C 85 C7  70 29 FA 20 90 BD CF 8D   .1ea.\..p). ....
2C B0 C6 0D 8D 65 69 25  E2 44 EA 63 20 8B 0F 31   ,....ei%.D.c ..1
20 A2 A6 0B 48 A4 0E 8B  C2 41 20 AC A8 7A C6 21    ...H....A ..z.!
57 9D 57 52 84 69 6D 80  0B 20 20 63 75 8D 87 BD   W.WR.im..  cu...
F8 49 09 0B C6 20 20 31  BD D2 C9 44 68 A0 20 CD   .I...  1...Dh. .
55 CE 5C 31 C7 31 CF 20  92 1C F7 F8 20 40 75 D3   U.\1.1. .... @u.
7A 20 20 5C 65 C4 00 A0  3C F7 20 65 20 EC 75 26   z  \e...<. e .u&
44 6E 0C 31 57 18 86 17  25 20 FB 9D 62 8D 6F 0D   Dn.1W...% ..b.o.
62 20 5A C2 74 2B 53 AC  20 3F 17 A5 25 A0 F2 62   b Z.t+S. ?..%..b
34 0B 97 15 62 F7 20 9C  00 20 9D C6 20 CC 00 CD   4...b. .. .. ...
56 65 E4 E7 6E 40 7A 4A  0C FF 65 04 68 FA 2B 1C   Ve..n@zJ..e.h.+.
75 6F 57 57 73 00 E0 FB  5C CD FA 1F C6 18 8D 0E   uoWWs...\.......
F7 87 87 0C 20 65 9D 9F  C2 E0 31 2B 20 20 F8 1A   .... e....1+  ..
```

Figure 4b

Encrypted Text using the Plain Text in Figure 4a. The first four bytes (80 04 00 00) define the Plain Text file length (00 00 04 80) in bytes. Usage: ENCRYPT ADFGVX.TXT GERMANS

```
20 20 20 20 20 20 20 20  46 69 72 73 74 20 6F 66          First of
20 61 6C 6C 2C 20 6C 65  74 27 73 20 67 65 6E 65   all, let's gene
72 61 74 65 20 61 20 73  74 72 69 6E 67 20 6F 66  rate a string of
20 72 65 70 65 61 74 65  64 20 63 68 61 72 61 63   repeated charac
74 65 72 73 3A 0D 0A 0D  0A 20 20 20 20 20 20 20  ters:....
20 20 20 20 20 20 20 78  78 78 78 78 78 78 78 78         xxxxxxxxx
78 78 78 78 78 78 78 78  78 78 78 78 78 78 78 78  xxxxxxxxxxxxxxxx
78 78 78 78 78 78 78 78  78 78 78 78 78 78 78 78  xxxxxxxxxxxxxxxx
78 78 78 78 78 78 78 78  78 78 78 0D 0A 20 20 20  xxxxxxxxxxx..
20 20 20 20 20 20 20 20  20 20 20 59 59 59 59 59             YYYYY
59 59 59 59 59 59 59 59  59 59 59 59 59 59 59 59  YYYYYYYYYYYYYYYY
59 59 59 59 59 59 59 59  59 59 59 59 59 59 59 59  YYYYYYYYYYYYYYYY
59 59 59 59 59 59 59 59  59 59 59 59 59 59 59 0D  YYYYYYYYYYYYYYY.
0A 0D 0A 20 20 20 20 20  20 20 20 20 54 68 65 73  ...         Thes
65 20 20 73 65 71 75 65  6E 63 65 73 20 20 77 69  e  sequences  wi
6C 6C 20 62 65 20 63 6F  6E 76 65 72 74 65 64 20  ll be converted
62 79 20 74 68 65 20 20  64 61 74 61 20 20 63 6F  by the  data  co
6D 70 72 65 73 73 69 6F  6E 20 0D 0A 20 20 20 20  mpression ..
20 20 20 20 20 72 6F 75  74 69 6E 65 20 74 6F 20       routine to
74 68 65 20 66 6F 6C 6C  6F 77 69 6E 67 20 74 68  the following th
72 65 65 20 63 68 61 72  61 63 74 65 72 20 72 65  ree character re
70 72 65 73 65 6E 74 61  74 69 6F 6E 3A 0D 0A 0D  presentation:...
0A 20 20 20 20 20 20 20  20 20 20 20 20 20 20 3C  .              <
78 3E 20 53 59 4E 20 3C  35 32 3E 20 3D 20 37 38  x> SYN <52> = 78
20 31 36 20 33 34 20 20  28 69 6E 20 68 65 78 29   16 34  (in hex)
0D 0A 20 20 20 20 20 20  20 20 20 20 20 20 20 20  ..
3C 59 3E 20 53 59 4E 20  3C 35 32 3E 20 3D 20 35  <Y> SYN <52> = 5
39 20 31 36 20 33 34 20  20 28 69 6E 20 68 65 78  9 16 34  (in hex
29 0D 0A 0D 0A 20 20 20  20 20 20 20 20 20 4E 6F  )....         No
74 69 63 65 20 20 74 68  61 74 20 77 65 20 61 72  tice  that we ar
65 20 6E 6F 74 20 72 65  73 74 72 69 63 74 65 64  e not restricted
20 74 6F 20 75 70 70 65  72 20 63 61 73 65 20 6C   to upper case l
65 74 74 65 72 73 20 20  61 6E 64 20 0D 0A 20 20  etters  and ..
20 20 20 20 20 20 20 64  69 67 69 74 73 20 20 6F         digits  o
6E 6C 79 2E 20 20 20 54  68 69 73 20 69 6D 70 6C  nly.   This impl
65 6D 65 6E 74 61 74 69  6F 6E 20 64 6F 65 73 20  ementation does
6E 6F 74 20 68 61 76 65  20 20 74 6F 20 20 72 65  not have  to  re
6D 6F 76 65 20 0D 0A 20  20 20 20 20 20 20 20 20  move ..
73 70 61 63 65 73 20 20  61 6E 64 20 70 75 6E 63  spaces  and punc
74 75 61 74 69 6F 6E 2E  20 20 20 54 68 75 73 2C  tuation.   Thus,
20 20 61 6C 6C 20 41 53  43 49 49 20 63 68 61 72    all ASCII char
61 63 74 65 72 73 20 28  30 20 20 74 6F 20 0D 0A  acters (0  to ..
20 20 20 20 20 20 20 20  20 31 32 37 29 20 63 61           127) ca
6E 20 62 65 20 75 73 65  64 2C 20 61 73 20 77 65  n be used, as we
6C 6C 20 61 73 20 74 68  65 20 62 79 74 65 20 76  ll as the byte v
61 6C 75 65 73 20 66 72  6F 6D 20 31 32 38 20 74  alues from 128 t
6F 20 32 35 35 2E 0D 0A  0D 0A 20 20 20 20 20 20  o 255.....
20 20 20 54 68 65 20 20  65 6E 63 72 79 70 74 65     The  encrypte
64 20 66 69 6C 65 20 77  69 6C 6C 20 6E 6F 74 20  d file will not
68 61 76 65 20 61 20 6F  6E 65 2D 74 6F 2D 6F 6E  have a one-to-on
65 20 63 6F 72 72 65 73  70 6F 6E 64 65 6E 63 65  e correspondence
20 0D 0A 20 20 20 20 20  20 20 20 20 62 65 74 77   ..         betw
65 65 6E 20 74 68 65 73  65 20 63 68 61 72 61 63  een these charac
74 65 72 73 20 61 6E 64  20 74 68 6F 73 65 20 70  ters and those p
72 6F 64 75 63 65 64 20  62 79 20 74 68 65 20 65  roduced by the e
6E 63 72 79 70 74 69 6F  6E 20 0D 0A 20 20 20 20  ncryption ..
20 20 20 20 20 70 72 6F  63 65 73 73 2C 20 20 69       process,  i
2E 65 2E 20 6C 65 74 74  65 72 20 66 72 65 71 75  .e. letter frequ
65 6E 63 79 20 61 6E 61  6C 79 73 69 73 20 77 69  ency analysis wi
6C 6C 20 6E 6F 74 20 68  65 6C 70 2E 20 20 54 68  ll not help.  Th
69 73 20 0D 0A 20 20 20  20 20 20 20 20 20 72 65  is ..         re
73 75 6C 74 20 69 73 20  64 75 65 20 74 6F 20 74  sult is due to t
68 65 20 74 72 61 6E 73  70 6F 73 69 74 69 6F 6E  he transposition
2C 20 20 77 68 69 63 68  20 6D 61 79 20 72 65 70  ,  which may rep
6C 61 63 65 20 61 6E 20  22 61 22 20 0D 0A 20 20  lace an "a" ..
20 20 20 20 20 20 20 69  6E 20 6F 6E 65 20 63 61         in one ca
73 65 20 62 79 20 61 20  22 59 22 20 61 6E 64 20  se by a "Y" and
69 6E 20 61 6E 6F 74 68  65 72 20 63 61 73 65 20  in another case
62 79 20 61 20 22 2A 22  2C 20 65 74 63 2E 0D 0A  by a "*", etc...
1A                                                .
```

Figure 4c

Decrypted Text using the Encrypted Text in Figure 4b.
Usage: DECRYPT ADFGVX.TXT GERMANS

then appending this information to the encrypted message.

In everyday use, this cipher is probably adequate. You can make it more secure by using the additions that I have suggested. Remember, however, that it all depends on the time value of the information that you are securing. If you want to be sure that unauthorized persons cannot decipher your information for decades, you should use a RSA or a DES cryptosystem. For shorter periods, this enhanced ADFGVX cipher should be effective: until the discovery of the DES and RSA methods, it was claimed to be one of the "best" ciphers around.

References

[1] C. E. Burton, "RSA: A Public Key Cryptography System, Parts 1 and 2," *DDJ*, March and April 1984.

[2] D. Kahn, *The Code Breakers*, Macmillan Book Co., 1967.

[3] C. Muller-Schloer, "A Microprocessor-based Cryptoprocessor," *IEEE MICRO*, vol. 3, no. 5, October 1983, pp. 5-15.

[4] C. C. Foster, *Cryptoanalysis for Microcomputers*, Hayden Book Co., 1982.

DDJ

(Listings begin on next page)

I would like to thank John K. Taber, the reviewer, for suggesting three improvements to this cipher system: a better random number generator, a more thorough data compression technique, and an alternate padding method.

Mr. Taber first pointed out that my pseudo-random number generator has a fairly small period (about 2,796,203) as pseudo-random number generators go. The short period makes possible a concerted attack on the cipher. Because there are 256! ways to fill the 16 × 16 array, and I am using only a small fraction of them (based on the period), he suggested a random number with a period of 2 ** 64, or possibly 2 ** 128, as being safe for the forseeable future. Note, however, that the ran( ) function is a stand-alone module and can be replaced by a random number generator of the user's choice, as long as the interface remains unchanged. The reader might be interested in knowing that Knuth has devoted an entire chapter to random numbers, their generation, and their testing in his book on seminumerical algorithms.

Other ways around this problem might be to use truly random physical phenomena to generate the random numbers. For example, the Allies during World War II used the X-System: a mercury vapor rectifier tube that generated wideband thermal noise. The output was sampled into six levels of equal probabilities at a 20 msec rate using nonuniform quantization. These random numbers, recorded onto plastic disks similar to phonograph records, were used to scramble voice messages. (See D. Kahn, "Cryptology and the Origins of Spread Spectrum," *IEEE Spectrum*, vol. 21, no. 9, September 1984, p. 74.)

But remember, you have to contend with the problem of key distribution: sending these random numbers, as well as the irrregular transposition key, to the receiver. The value of the message and its life time will dictate the extremes to which you must go to protect your data.

Mr. Taber next suggested that better data compression schemes exist. Because I used this cipher in particular to encrypt data files that consisted of numerous repeated character strings, the run length code suited my purposes. As I pointed out in the article, Mr. Greenlaw has an excellent, public domain data compression package (SQ/USQ). Readers interested in Huffman compression should obtain a copy of Mr. Greenlaw's source code from your local RCP/M bulletin board system. You could certainly use it to enhance my compression scheme.

One reason that I did not use the Huffman algorithm was because you have to scan the text twice to obtain the compression. You use the first scan to obtain the frequency information and to assign the compression codes. The second scan compresses the data using the compression codes and saves the result on disk. To uncompress the data, you must retain the character/compression code substitution pairs (or encrypt the compression codes along with the compressed data). If the codes are stored in a known position, they could be used to break the cipher. On the other hand, because these Huffman characters are variable length codes, they might make an attack on the cipher more difficult. Anyway, I did not have the need to go to such lengths.

Finally, Mr. Taber suggested an alternate padding method. He recommended the use of an array at least as long as the padding needed, say, 256 bytes. The pad array is filled with some known pattern (byte by byte). Then each byte of the clear text is read and exclusive or'ed with the next element of the pad array in a circular fashion; i.e., when the 256th byte is used, the process starts over with the first byte. When padding is required, the pad bytes are drawn from the pad array. This method, or another comparable method, avoids the weakness of having a known range of bytes or known bytes in the final block.

I do not believe that most people will have to make any of these modifications; the cipher (as written) should be safe enough for most uses. For example, MSDOS/PCDOS, unlike MP/M-II and CP/M-3, does not allow for making files read-only. Thus, users could use this cipher to encrypt those sensitive personnel files contained on a PC/XT that is used by several employees.

With respect to irregular transposition, Mr. Taber pointed out that C. A. Deavours, editor of *Cryptologia*, has written a BASIC program to help determine the cipher "breakpoints" (i.e., the possible tops and bottoms for given irregular transposition dimensions). The program uses an "odds in favor" approach from statistics to measure when the right breakpoints are determined. Deavours claims that he can break the cipher within five to ten minutes using this technique—starting from scratch. (Deavours' article describing the technique, along with the program, appears in *Cryptologia*, vol. 5, no. 4, October 1981, pp. 247-251.) However, when you combine the irregular transposition cipher with the enhanced ADFGVX cipher, the job of breaking the code becomes orders of magnitude more difficult!

 (Text begins on page 48)

Listing One

```
/*######################################################################
###                                                                 ###
###       Copyright 1983, Charles E. Burton, Denver, Colorado       ###
###                                                                 ###
### All rights reserved.  Permission granted to use this software for ###
###   personal, non-commercial purposes only.                       ###
###                                                                 ###
######################################################################*/

/*
 * PROGRAM NAME:   SHUFFLE.C
 * PURPOSE:   Card shuffling routine --
 *                 Shuffles a "deck" of cards.  The number of cards is NOCARDS.
 *            Random number routine --
 *                 Generates a random interger
 *
 * LANGUAGE:   C
 * AUTHOR:   CEB
 * USAGE:   SHUFFLE(POSITION,VALUE)
 *               POSITION [I] -- returns as the position of Card #I in the Deck.
 *               VALUE [J] -- returns as the value of the Jth Card in the Deck.
 *          RAN(MAX)
 *               MAX -- integer specifying the maximum random integer that can
 *                      be returned.
 * ARRAYS USED:   POSITION[NOCARDS],VALUE[NOCARDS]
 * EXTERNALS:
 * UPDATE HISTORY:   INITIAL RELEASE -- 11/25/83 CEB
 */

/* Card shuffling routine */

shuffle(position,value)
     char position[]; /* POSITION [NOCARDS] on entry */
     char value[]; /* VALUE [NOCARDS] on entry */

     {
     char temp;
     int i,j;

     for (i=0; i < NOCARDS; i++) /* initialize CARDS values */
          value [i] = i;
     for (i=0; i < NOCARDS; i++) /* shuffle */
          {
          j = ran(NOCARDS); /* swap index */
          temp = value [i]; /* swap CARDS values */
          value [i] = value [j];
          value [j] = temp;
          }
     for (i=0; i < NOCARDS; i++) /* get positions of cards */
          position [ value [i] ] = i;
     }

/*********************************************************************/

/*
   Random number generator --
   adapted from the FORTRAN version
   in "Software Manual for the Elementary Functions"
   by W.J. Cody, Jr and William Waite.
 */

ran(max)
     int max;

     {
     static long int iy = 100001;

     iy *= 125;
     iy -= (iy/2796203) * 2796203;
     return ((int) (max * iy / 2796203));
     }
```

End Listing One

(Listing Two begins on page 60)

Powerful in circuit emulation, priced well within your grasp. That's NICE.™

NICE may be only 3" square and ½" thick, but it hands you full speed, real-time emulation—over 50 emulation functions, software breakpoints, all memory addresses and all I/O ports.

Just plug NICE directly into the target MP socket and any RS232 terminal for system development, troubleshooting, debugging or testing . . . at home, in the lab or in the field.

And NICE hands you all this performance, portability and versatility for only $498*. . . the best emulator price/performance ratio on the market, hands down.

Call in your order today using your VISA or Mastercard number: (800) NICOLET outside CA, or (415) 490-8300 in CA. Or send your check or money order to NICE, Nicolet Paratronics Corporation, 201 Fourier Avenue, Fremont, CA 94539.

*Payment by check, money order, VISA or MasterCard.

NICE is a trademark of Nicolet Paratronics Corporation

®Z80 is a trademark of Zilog, Inc.

NIC Nicolet

Circle **no. 60** on reader service card.

DSD 80

FULL SCREEN SYMBOLIC DEBUGGER

"THE SINGLE BEST DEBUGGER FOR CP/M-80. A TRULY AMAZING PRODUCT."

LEOR ZOLMAN
AUTHOR OF BDS C

- ☐ Complete Upward Compatibility with DDT
- ☐ Simultaneous instruction, register, stack & memory displays
- ☐ Software in-circuit-emulator provides write protected memory, execute only code and stack protection.
- ☐ Fifteen single keystroke commands
- ☐ Thirty day money back guarantee
- ☐ On-line help & 50 page user manual

ONLY $195.

IBM PC VERSION AVAILABLE SOON!

SOFTADVANCES

P.O. BOX 49473 AUSTIN, TEXAS 78765 (512) 478-4763

MasterCard VISA

Circle **no. 63** on reader service card.

DeSmet C

8086/8088 Development Package **$109**

FULL DEVELOPMENT PACKAGE
- Full **K&R** C Compiler
- Assembler, Linker & Librarian
- Full-Screen Editor
- Execution Profiler
- Complete **STDIO** Library (>120 Func)

Automatic DOS 1.X/2 X SUPPORT

BOTH 8087 AND S/W FLOATING POINT

OVERLAYS

OUTSTANDING PERFORMANCE
- First and Second in AUG '83 BYTE benchmarks

SYMBOLIC DEBUGGER **$50**

- Examine & change variables by name using C expressions
- Flip between debug and display screen
- Display C source during execution
- Set multiple breakpoints by function or line number

DOS LINK SUPPORT **$35**

- Uses DOS .OBJ Format
- LINKs with DOS ASM
- Uses Lattice® naming conventions

Check: ☐ Dev. Pkg (109)
☐ Debugger (50)
☐ DOS Link Supt. (35)

SHIP TO: ______________________

____

____

C WARE CORPORATION

P.O. BOX C
Sunnyvale, CA 94087
(408) 720-9696

All orders shipped UPS surface on IBM format disks. Shipping included in price. California residents add sales tax. Canada shipping add $5. elsewhere add $15. Checks must be on US Bank and in US Dollars. Call 9 a.m. – 1 p.m. to CHARGE by VISA/MC/AMEX.

Circle **no. 18** on reader service card.

Listing Two

```
/*########################################################################
###                                                                   ###
###         Copyright 1983, Charles E. Burton, Denver, Colorado       ###
###                                                                   ###
### All rights reserved.  Permission granted to use this software for ###
###   personal, non-commercial purposes only.                         ###
###                                                                   ###
########################################################################*/

/*
 * PROGRAM NAME:  PWSORT.C
 * PURPOSE:   Password sorting routine --
 *            Sorts password in largest to smallest character ordeer.  Used to
 *            specify the column order to pull out columns in TRANSPOSE() and
 *            UNTRANSPOSE().
 *
 * LANGUAGE:  C
 * AUTHOR:   CEB
 * USAGE:   PWSORT(PASSWORD)
 *                PASSWORD -- character array containing the encryption/
 *                            decryption key.
 *
 * ARRAYS USED:  PASSWORD[]
 * EXTERNALS:
 * UPDATE HISTORY:   INITIAL RELEASE -- 11/25/83 CEB
 */

#define min(x,y) ((x) < (y) ? (x) : (y))

pwsort(password)
     char password[];

     {
     int i,j,temp;

     pwlen = strlen(password); /* get length of PASSWORD string */
     if (pwlen < MINPWLEN) /* PASSWORD too short */
          {
          printf("\n*** Password:  %s is too short ( < 5 characters ) ***\n",
                 password);
          exit(0);
          }
     pwlen = min(pwlen,MAXPWLEN); /* truncate PASSWORD if necessary */
     for (i = 0; i < pwlen; i++) /* initialize Password Column Order */
          pwcolumnorder[i] = i;
     for (i = 0; i < pwlen - 1; i++) /* sort Password by Column Order */
          for (j = i + 1; j < pwlen; j++)
               if (password[pwcolumnorder[i]] < password[pwcolumnorder[j]])
                                         /* largest to smallest character order */
                    {
                    temp = pwcolumnorder[i];
                    pwcolumnorder[i] = pwcolumnorder[j];
                    pwcolumnorder[j] = temp;
                    }
     }
```

End Listing Two

Listing Three

```
/*########################################################################
###                                                                   ###
###         Copyright 1983, Charles E. Burton, Denver, Colorado       ###
###                                                                   ###
### All rights reserved.  Permission granted to use this software for ###
###   personal, non-commercial purposes only.                         ###
###                                                                   ###
########################################################################*/

/*
 * PROGRAM NAME:  ENCRYPT.C
 * PURPOSE:   Encryption using the ADFGVX Cipher --
 *            re. C.C. Foster, "Cryptoanalysis for Microcomputers," Hayden Book
 *            Co. (Rochelle Park, NJ), p. 222.
```

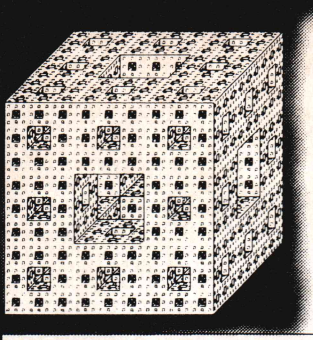
```
 *
 *
 * LANGUAGE:  C
 * AUTHOR:  CEB
 * USAGE:  ENCRYPT <filename> <password>
 *              <filename> -- File Name to be encrypted
 *              <password> -- 5 to 10 character key to be used for encryption
 *
 * ARRAYS USED:  FILENAME[20],ENCFNAME[20],PASSWORD[11],BUF1[110],BUF2[110],
 *               POSITION[NOCARDS],VALUE[NOCARDS],PWCOLUMNORDER[MAXPWLEN]
 * EXTERNALS:  PWSORT(), SHUFFLE(), RAN()
 * UPDATE HISTORY:  INITIAL RELEASE -- 11/25/83 CEB
 */

#include "stdio.h"

#define NOCARDS 256

#define NO 0
#define YES 1

main(argc,argv)
     char *argv[];
     int argc;

     {
     static char filename[20],encfname[20],password[11],*str;
     static char c,buf1[110],buf2[110],position[NOCARDS],value[NOCARDS];
     int i,fnlen,pwlen,blklen,idx,
         done = NO;
     long int ftell(),dummy;
     union
          {
          long int xlong;
          char xbyte[sizeof(dummy)];
          } filelen;
     FILE *fopen(),*fpin,*fpout;

  if (argc != 3)
       {
       printf("\nUsage:  ENCRYPT <filename> <password>\n");
       exit(0);
       }
  for (i=0, idx=sizeof(filename), str=argv[1];  /* get file name to use */
       *str != '\0' && i < 14; i++, str++)
       {
       filename[i]=encfname[i]=*str;  /* get file name */
       if (*str == '.')  /* start of <ext> found? */
            idx=i+1;  /* get index to start of <ext> */
       }
  filename[i]=encfname[i]='\0';  /* terminate file name */
  fnlen=i;  /* get length of file name */
  switch (fnlen-idx)  /* based on length of <ext> */
       {
       case 1:
            /* only one character in <ext> */
            encfname[idx+2]='\0';  /* move EOS over one character */
            fnlen++;  /* account for added character */
       case 2:
       case 3:
            /* two or three characters in <ext> */
            encfname[idx+1]='Y';  /* make 2nd character of <ext> a 'Y' */
            break;
       default:
            if (idx == fnlen)  /* no <ext>, but '.' exists */
                 strcpy(encfname+fnlen,"YYY");  /* add <ext> */
            else if (idx == sizeof(filename))  /* no '.' and no <ext>? */
                 strcpy(encfname+(fnlen++),".YYY");  /* add <ext> */
            else  /* invalid file name */
                 {
                 printf("\n*** Bad <filename>:  %s ***\n",filename);
                 exit(0);
                 }
            fnlen += 3;  /* account for added "YYY" */
            break;
       }
  for (i=0, str=argv[2]; *str != '\0' && i < sizeof(password)-1; i++)
       password[i]=*(str++);  /* get Password */
  password[i]='\0';  /* terminate Password */
  pwlen=i;  /* get length of Password */
```

(Continued on next page)

Listing Three

```
blklen=pwlen*pwlen+pwlen/2;   /* get length of block to read/write for
                                 file I/O to get irregular columns for
                                 password */
if (blklen != 2*(blklen/2))   /* BLKLEN odd? */
      blklen++;   /* make it even */
if ((fpin=fopen(filename,"r")) == NULL)   /* cannot open input file? */
      {
      printf("\n*** Cannot open %s ***\n",filename);
      exit(0);
      }
if ((fpout=fopen(encfname,"w")) == NULL)   /* cannot open output file? */
      {
      printf("\n*** Cannot open %s ***\n",encfname);
      fclose(fpin);   /* close input file */
      exit(0);
      }
fseek(fpin,0L,2);   /* find end of file */
filelen.xlong=ftell(fpin);   /* get length of plain text file */
fclose(fpin);   /* close the input file */
fpin=fopen(filename,"r");   /* re-open the input file */
for (i=0; i < sizeof(filelen.xlong); i++)
      putc(filelen.xbyte[i],fpout);   /* save file length at beginning of
                                         output file */
shuffle(position,value);   /* randomize cipher character set */
pwsort(password);   /* get transposition for row/column pairs */
do   /* encrypt plain text file */
      {
      for (i=0; i < blklen; i++)   /* read a block of plain text */
            {
            if ((c=compress(fpin)) == EOF)   /* Compress text, EOF found? */
                  {
                  done=YES;   /* indicate last pass of plain text */
                  if (i)   /* at least one value in BUF1 */
                        for ( ; i < blklen; i++)   /* fill in rest of BUF1 */
                              {
                              c=ran(NOCARDS);   /* generate a random char */
                              buf1[i++]=position[c]/16;
                                                /* char row position */
                              buf1[i]=position[c]%16;
                                                /* char column position */
                              }
                  break;
                  }
            /* fill BUF1 */
            buf1[i++]=position[c]/16;   /* char row position */
            buf1[i]=position[c]%16;   /* char column position */
            }
      if (i) /* BUF1 full? */
            {
            transpose(buf1,buf2,blklen);   /* encrypt using Password */
            for (i=0; i < blklen; i += 2)   /* write a block of encrypted
                                               text */
                  putc(value[16*buf2[i]+buf2[i+1]],fpout);
            }
      }
while (done == NO);
fclose(fpin);   /* close files */
fclose(fpout);
if ((fpout=fopen(filename,"r+")) == NULL)   /* cannot open output file? */
      {
      printf("\n*** Cannot open %s ***\n",filename);
      exit(0);
      }

      else   /* overwrite all records so that they cannot be recovered */
            {
            while (filelen.xlong--)   /* Null out plain text file */
                  putc(0xF6,fpout);   /* use formatting data character to null */
            fclose(fpout);   /* close the file */
            }
      unlink(filename);   /* erase plain text file */
      printf("\nEncryption completed\n");
      }
```

(Continued on page 64)



WALTZ LISP(TM)

*The one and only **adult** Lisp system for CP/M users.*

Waltz Lisp is a very powerful and complete implementation of the Lisp programming language. It includes features previously available only in large Lisp systems. In fact, Waltz is substantially compatible with *Franz* (the Lisp running under *Unix*), and is similar to *MacLisp*. Waltz is perfect for Artificial Intelligence programming. It is also most suitable for general applications.

Much faster than other microcomputer Lisps. • Long integers (up to 611 digits). Selectable radix • True dynamic character strings. Full string operations including fast matching/extraction. • Flexibly implemented random file access. • Binary files. • Standard CP/M devices. • Access to disk directories. • Functions of type lambda (expr), nlambda (fexpr), lexpr, macro. • Splicing and non-splicing character macros. • User control over all aspects of the interpreter. • Built-in prettyprinting and formatting facilities. • Complete set of error handling and debugging functions including user programmable processing of undefined function references. • Virtual function definitions. • Optional automatic loading of initialization file. • Powerful CP/M command line parsing. • Fast sorting/merging using user defined comparison predicates. • Full suite of mapping functions, iterators, etc. • Assembly language interface. • Over 250 functions in total. • The best documentation ever produced for a micro Lisp (300+ full size pages, hundreds of illustrative examples).

Waltz Lisp requires CP/M 2.2, Z80 and 48K RAM (more recommended). All common 5'' and 8'' disk formats available.

ProCode (TM) INTERNATIONAL
15930 SW Colony Pl.
Portland, OR 97224

Unix* Bell Laboratories.
CP/M* Digital Research Corp.

Version 4.4 (Now includes Tiny Prolog written in Waltz Lisp.)

$169*

*Manual only: $30 (refundable with order). All foreign orders: add $5 for surface mail, $20 for airmail. COD add $3. Apple CP/M and hard sector formats add $15.

Call free **1-800-LIP-4000** Dept. #11
In Oregon and outside USA call 1-503-684-3000

Circle **no. 73** on reader service card.

THE PROGRAMMER'S SHOP™

helps compare evaluate and find products. Get answers.

SERVICE: FREE LITERATURE
One free call covers all programmer's software. Ask for a ''Packet'' on: ''AI'', BASIC, C, COBOL, Debuggers, Editors, FORTH, FORTRAN, Libraries, PASCAL, UNIX/PC or 30 ''addons'' for ''C''.

"C" Language

	OUR PRICE
MSDOS: C86 - 8087, reliable	call
Lattice 2.1 - improved - 30 addons	call
Microsoft C2.x	329
Williams - debugger	call
Instant C Interpreter, fast, full, debug	500
CPM80: Ecosoft C-now solid, full, faster	225
MAC: **Megamax** - fast, full, tight	295

EDITORS

	Runson	
BRIEF - Intuitive, flexible	PCDOS	195
PMATE - powerful	8086	195
VEDIT - full, liked	8086	119

ARTIFICIAL INTELLIGENCE

IQ LISP - full 1000K RAM	PCDOS	call
TLC LISP - with ''classes'', nice	MSDOS	250
MicroProlog - by Logic Prog. Assem.	MSDOS	285
PROLOG-86 - standard, Learn fast	MSDOS	125
EXSYS - Expert System	PCDOS	295

Feature

PERISCOPE DEBUGGER — ''Reset Box'', own RAM, Registers, symbols, line nums, 2 screen, PCDOS. $295.

Recent Discovery

Introducing-C : C Interpreter and training system. Nice. Thorough. PCDOS. Only $95.

FORTRAN

	Runson	OUR PRICE
MS Fortran - Improved	MSDOS	249
DR Fortran-86 - full '77'	8086	259
F77L - by Leahy - Nice.	MSDOS	449
RM Fortran	MSDOS	545

SUPPORT PRODUCTS

LIBRARIES: BTRIEVE ISAM	MSDOS	215
CIndex+ - ISAM, source, no royalt.	8086	375
CSHARP Realtime - source, full	MSDOS	600
CUtil by Essential	MSDOS	139
DATABURST - Screens-C, BAS	MSDOS	215
GraphiC - 4200 x 3100, source	MSDOS	195
Greenleaf C - thorough	MSDOS	165
HALO Graphics - fast, full	PCDOS	175
TOOLS: MULTILINK - multitask	PCDOS	65
Polylibrarian-thorough	MSDOS	89
PolyMAKE-manage, compiles	PCDOS	89
Profiler-86-easy to setup, symbols	MSDOS	125
XENIX - ''true S3'', rich, + C-MSDOS	PC	1285

Call for a catalog and *solid value*
800-421-8006
THE PROGRAMMER'S SHOP™
128-D Rockland Street, Hanover, MA 02339
Visa Mass: 800-442-8070 or 617-826-7531 MasterCard

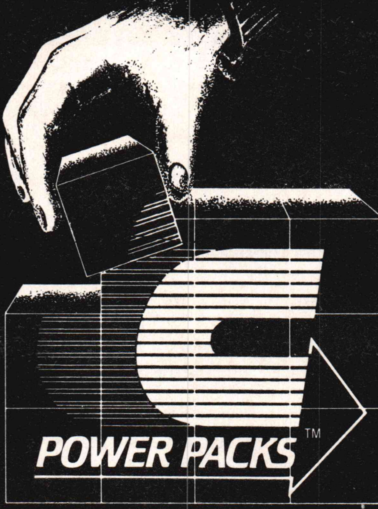
Note: All prices subject to change without notice.

Mention this ad. Some prices are specials.

All formats available.
Ask about POs, COD.

Circle **no. 67** on reader service card.

''This is a beautifully documented, incredibly comprehensive set of C Function Libraries.''
— Dr. Dobb's Journal



COMPLETE SOURCES

- **PACK I: Building Blocks I** $149
  250 Functions: DOS, Printer, Video, Asynch
- **PACK 2: Database** $399
  100 Functions: B-Trees, Variable Records
- **PACK 3: Communications** $149
  135 Functions: Smartmodem™, Xon/Xoff, Modem-7, X-Modem
- **PACK 4: Building Blocks II** $149
  100 Functions: Dates, Text Windows, Pull-down Menus Data Compression
- **PACK 5: Mathematics I** $99
  35 Functions: Log, Trig, Square Root
- **PACK 6: Utilities I** $99
  Archive, Diff, Replace, Scan, Wipe (Executable Files only)

Lattice™, Microsoft™, DeSmet™, CI-86™ Compilers on IBM PC/XT/AT™
Small and Large Memory Models.
Credit cards accepted
($7.00 handling/Mass. add 5%)

SOFTWARE HORIZONS INC.
165 Bedford Street
Burlington, Mass. 01803
(617) 273-4711

NOVUM ORGANUM

Circle **no. 90** on reader service card.

ADFGVX Cipher System (Listing Continued, text begins on page 48)

Listing Three

```
/***************************************************************************/

/* Multiple Character Compression.  Encodes repeated characters. The Stream is
   byte for byte pass-through except that SYN is encoded as SYN/0 and repeated
   byte values are encoded as Byte/SYN/Count, where Count > = 3.  This routine
   is  a state machine representation and is modeled after the GETNCR()
   function in SQ.C (squeezer) by Richard Greenlaw.
*/

#define NOHISTORY 0        /* do not consider previous input */
#define SENTCHAR  1        /* LASTCHAR sent, no lookahead yet */
#define SENDNEWC  2        /* NEWCHAR sent, previous sequence done */
#define SENDCNT   3        /* NEWCHAR sent, SYN sent, send COUNT next */

#define SYN 0X16           /* duplicate character indicator */

compress(stream)
     FILE *stream;

     {
     static char state = NOHISTORY; /* states */
     static int count; /* Count of consecutive identical characters */
     static int lastchar,nextchar;

     switch (state)
          {
          case NOHISTORY:
               /* no relevant history */
               state = SENTCHAR;
               return (lastchar = getc(stream));
               break;
          case SENTCHAR:
               /* LASTCHAR is sent, need to lookahead */
               switch (lastchar)
                    {
                    case SYN:
                         /* actual SYN character found */
                         state = NOHISTORY;
                         return (0);
                         break;
                    case EOF:
                         /* end of file found */
                         return (lastchar);
                         break;
          default:
               /* any other character found */
               for (count=1; (nextchar = getc(stream)) == lastchar &&
                    count < 255; count++) /* count like characters */
                    ;
               switch (count)
                    {
                    case 1:
                         /* one character found */
                         return (lastchar = nextchar);
                         break;
                    case 2:
                         /* two characters found */
                         state = SENDNEWC;
                         return (lastchar);
                         break;
                    default:
                         /* three or more characters found */
                         state = SENDCNT;
                         return (SYN);
                         break;
                    }
               break;
          }
 case SENDNEWC:
      /* previous sequence complete, send NEWCHAR */
      state = SENTCHAR;
      return (lastchar = nextchar);
      break;
 case SENDCNT:
```

```
            /* sent SYN for repeat sequence, send COUNT */
            state = SENDNEWC;
            return (count);
            break;
      default:
            printf("\n*** Bad STATE in COMPRESS() ***\n");
            exit(0);
            break;
      }
}
/***************************************************************************/

/* Transpose via Columns based on Password.  This routine takes a string and
   transposes the characters to column order based on a password. The password
   length is the width of the table and the password has been sorted to
   specify the column order to pull out the columns. */

#define MAXPWLEN 10
#define MINPWLEN  5

static int pwlen,pwcolumnorder[MAXPWLEN];

#include "PWSORT.C"

transpose(from,to,len)
      char from[],to[];
      int len;

      {
      int i,j,k;

      i = 0;
      for (j = 0; j < pwlen; j++) /* transpose in column order */
            for (k = pwcolumnorder[j]; k < len; k += pwlen)
                                                 /* transpose characters */
                  to[i++] = from[k];
      }

#include "SHUFFLE.C"
```

End Listing Three

Listing Four

```
/*##########################################################################
###                                                                      ###
###          Copyright 1983, Charles E. Burton, Denver, Colorado         ###
###                                                                      ###
### All rights reserved.  Permission granted to use this software for   ###
###   personal, non-commercial purposes only.                            ###
###                                                                      ###
##########################################################################*/

/*
 * PROGRAM NAME:  DECRYPT.C
 * PURPOSE:  Decryption using the ADFGVX Cipher --
 *           re. C.C. Foster, "Cryptoanalysis for Microcomputers," Hayden Book
 *           Co. (Rochelle Park, NJ), p. 222.
 *
 *
 * LANGUAGE:  C
 * AUTHOR:  CEB
 * USAGE:  DECRYPT <filename> <password>
 *              <filename> -- File Name to be decrypted
 *              <password> -- 5 to 10 character key to be used for decryption
 *
 * ARRAYS USED:  FILENAME[20],ENCFNAME[20],PASSWORD[11],BUF1[110],BUF2[110],
 *               POSITION[NOCARDS],VALUE[NOCARDS],PWCOLUMNORDER[MAXPWLEN]
 * EXTERNALS:  PWSORT(), SHUFFLE()
 * UPDATE HISTORY:  INITIAL RELEASE -- 11/25/83 CEB
 */

#include "stdio.h"

#define NOCARDS 256

#define NO 0
#define YES 1
```

(Continued on next page)

Listing Four

```
static char password[11],position[NOCARDS],value[NOCARDS];
static int blklen;

main(argc,argv)
     char *argv[];
     int argc;

     {
     static char filename[20],decfname[20],*str,c;
     int i,fnlen,pwlen,idx;
     long int dummy;
     union
          {
          long int xlong;
          char xbyte[sizeof(dummy)];
          } filelen;
     FILE *fopen(),*fpin,*fpout;

     if (argc != 3)
          {
          printf("\nUsage:  DECRYPT <filename> <password>\n");
          exit(0);
          }
     for (i=0, idx=sizeof(filename), str=argv[1];  /* get file name to use */
          *str != '\0' && i < 14; i++, str++)
          {
          filename[i]=decfname[i]=*str;  /* get file name */
          if (*str == '.')  /* start of <ext> found? */
               idx=i+1;  /* get index to start of <ext> */
          }
     filename[i]=decfname[i]='\0';  /* terminate file name */
     fnlen=i;  /* get length of file name */
     switch (fnlen-idx)  /* based on length of <ext> */
          {
          case 1:
               /* only one character in <ext> */
               decfname[idx+2]='\0';  /* move EOS over one character */
               fnlen++;  /* account for added character */
          case 2:
          case 3:
               /* two or three characters in <ext> */
               decfname[idx+1]='Y';  /* make 2nd character of <ext> a 'Y' */
               break;
          default:
               if (idx == fnlen)  /* no <ext>, but '.' exists */
                    strcpy(decfname+fnlen,"YYY");  /* add <ext> */
               else if (idx == sizeof(filename))  /* no '.' and no <ext>? */
                    strcpy(decfname+(fnlen++),".YYY");  /* add <ext> */
               else  /* invalid file name */
                    {
                    printf("\n*** Bad <filename>:  %s ***\n",filename);
                    exit(0);
                    }
               fnlen += 3;  /* account for added "YYY" */
               break;
          }
     for (i=0, str=argv[2]; *str != '\0' && i < sizeof(password)-1; i++)
          password[i]=*(str++);  /* get Password */
     password[i]='\0';  /* terminate Password */
     pwlen=i;  /* get length of Password */
     blklen=pwlen*pwlen+pwlen/2;  /* get length of block to read/write for file
                                     I/O to get irregular columns for both
                                     passwords */
     if (blklen != 2*(blklen/2))  /* BLKLEN odd? */
          blklen++;  /* make it even */
     if ((fpin=fopen(decfname,"r")) == NULL)  /* cannot open input file? */
          {
          printf("\n*** Cannot open %s ***\n",decfname);
          exit(0);
          }
      if ((fpout=fopen(filename,"w")) == NULL)  /* cannot open output file? */
           {
           printf("\n*** Cannot open %s ***\n",filename);
```

(Continued on page 68)

Listing Four

```
            fclose(fpin);   /* close input file */
            exit(0);
            }
      for (i=0; i < sizeof(filelen.xlong); i++)
            filelen.xbyte[i]=getc(fpin);   /* get original file length */
      shuffle(position,value);   /* randomize cipher character set */
      pwsort(password);   /* get transposition for row/column pairs */
      while (((c=expand(fpin)) != EOF) && (filelen.xlong--))
                                               /* decrypt & expand the file */
            putc(c,fpout);   /* write plain text */
      fclose(fpin);   /* close files */
      fclose(fpout);
      printf("\nDecryption completed\n");
      }

/**********************************************************************/

/* Multiple Character Expansion.  Decodes compressed characters. The Stream
   is byte for byte pass-through except that SYN/0 is decoded as SYN and
   Byte/SYN/Count is decoded as Byte repeated Count times.  This routine
   is a state machine representation and is modeled after the GETNCR()
   function in SQ.C (squeezer) by Richard Greenlaw.
*/

#define NOHISTORY 0         /* do not consider previous input */
#define SENTCHAR  1         /* LASTCHAR sent, no lookahead yet */
#define SENDRPT   2         /* LASTCHAR sent, SYN found, found COUNT, repeat
                               LASTCHAR */

#define SYN 0X16            /* duplicate character indicator */

expand(stream)
      FILE *stream;

      {
      static char state = NOHISTORY; /* states */
      static int count; /* Count of consecutive identical characters */
      static int lastchar,nextchar;

      switch (state)
            {
            case NOHISTORY:
                  /* no relevant history */
                  switch (lastchar = getcnxt(stream))
                        {
                        case SYN:
                              /* must be a SYN character, check */
                              switch (getcnxt(stream))
                                    {
                                    case 0:
                                          /* SYN character found */
                                          return (lastchar);
                                          break;
                                 default:
                                       /* bad SYN character found */
                                       printf("\n*** Bad SYN in EXPAND() ***\n");
                                       exit(0);
                                       break;
                                 }
                           break;
                     default:
                           /* another character found */
                           state = SENTCHAR;
                           return (lastchar);
                           break;
                     }
               break;
         case SENTCHAR:
               /* LASTCHAR is sent, need to lookahead */
               switch (lastchar)
                     {
                     case EOF:
                           /* end of file found */
                           return (lastchar);
```

```
		break;
	default:
		/* any other character found */
		switch (nextchar = getcnxt(stream))
			{
			case SYN:
				/* repeating characters found */
				switch (count = getcnxt(stream))
					{
					case 0:
						/* SYN character found */
						state = NOHISTORY;
						return (nextchar);
						break;
					default:
						/* actual COUNT found */
						state = SENDRPT;
						count -= 2; /* adjust for 2 sent
						               characters */
						return (lastchar);
						break;
					}
				break;
			default:
				/* another character found */
				return (lastchar = nextchar);
				break;
			}
		break;
	}

			case SENDRPT:
				/* repeat sequence, send LASTCHAR COUNT times */
				if (count > 1)
					{
					/* keep sending LASTCHAR */
					count--;
					return (lastchar);
					}
				else if (count == 1)
					{
					/* final LASTCHAR */
					state = NOHISTORY;
					return (lastchar);
					}
				else
					{
					/* problem with COUNT */
					printf("\n*** Bad COUNT in EXPAND() ***\n");
					exit(0);
					}
				break;
			default:
				printf("\n*** Bad STATE in EXPAND() ***\n");
				exit(0);
				break;
			}
		}

/***************************************************************************/

/* Get next character from input stream.  Must read a block of data,
   untranspose it and distribute the block a character at a time. */

getcnxt(stream)
	FILE *stream;

	{
	static char buf1[110],buf2[110],c;
	int i;
	static int done = NO,
	           bufidx = sizeof(buf1);

	if ((bufidx >= blklen) && (done == NO))
```

(Continued on next page)

Listing Four

```
                        /* need to read more characters into BUF1? */
{
for (i=0; i < blklen; i++)   /* read a block of encrypted text */
     {
     if ((c=getc(stream)) == EOF)   /* EOF found? */
          {
          done=YES;   /* indicate last pass of plain text */
          break;
          }
     /* fill BUF1 */
     buf1[i++]=position[c]/16;   /* char row position */
     buf1[i]=position[c]%16;   /* char column position */
     }

          if (done == NO)   /* last Block meaningless, ignore it? */
               {
               untranspose(buf1,buf2,blklen);   /* decrypt with Password */
               bufidx=0;   /* reset Block Index to start of BUF1 */
               }
          }
     if (bufidx < blklen)   /* characters still available? */
          {
          bufidx += 2;   /* point to next row/column pair */
          return (value[16*buf2[bufidx-2]+buf2[bufidx-1]]);
                                        /* return a decrypted character */
          }
     else if (done == YES)   /* no more characters available & EOF? */
          return (EOF);   /* indicate done */
     else   /* character stream problem */
          {
          printf("\n*** Synchronization error in GETCNXT() ***\n");
          exit(0);
          }
     }

/***************************************************************************/

/* Untranspose via Columns based on Password.  This routine takes a string and
   transposes the characters to row order based on a password. The password
   length is the width of the table and the password has been sorted to
   specify the column order to put back the columns before pulling out rows. */

#define MAXPWLEN 10
#define MINPWLEN  5

static int pwlen,pwcolumnorder[MAXPWLEN];

#include "PWSORT.C"

untranspose(from,to,len)
     char from[],to[];
     int len;

     {
     int i,j,k;

     i = 0;
     for (j = 0; j < pwlen; j++) /* transpose in column order */
          for (k = pwcolumnorder[j]; k < len; k += pwlen)
                                        /* transpose characters */
               to[k] = from[i++];
     }

#include "SHUFFLE.C"
```

End Listings

More dBASE Tips and Techniques

by Gene Head

I wrote a dBASE II command file called DB-SQZ.CMD that will squeeze and tokenize any command file just as DBCODE does in Ashton-Tate's RUNTIME package. My customers who already owned dBASE II could run these "scrambled" command files and not have to purchase RUNTIME. (DB-SQZ.CMD may appear in a future column or call me for a modem download.)

Unfortunately, every customer that owned dBASE II also owned an IBM PC. In their wisdom Ashton-Tate made the default command file extension different for the MSDOS/IBM version of dBASE II. CP/M command files are of the type .CMD and IBM command files are .PRG. So I have this wonderful program that protects my command files from tampering, but I can't use it unless I change every file type from .CMD to .PRG on every disk!

There should be some way to keep the filenames as they are and make dBASE II change what it looks for when it meets a DO FILENAME.

The best solution was to make dBASE II look for the file type I told it to look for; that is, have my CP/M dBASE II look for .PRG instead of .CMD command files.

This is easy enough because the file type is stored as plain ASCII text and easily patched with the POKE command. I wrote a simple command file called FLIP.IT that changed the default file type using the POKE function.

But wait. Now I want to go the other direction! Make the IBM dBASE II program look for .CMD instead of .PRG command files. Furthermore, I need each direction for several versions (2.4, 2.41, 2.41 Z80) of dBASE II. This could get confusing.

Finally, I came up with the idea of writing the program in the listing (page 72) called MAKEFLIP.XXX that will "write" a customized version of FLIP.IT for any version of dBASE II almost automatically! Now, I only need to have a copy of MAKEFLIP.XXX when I go to work in any dBASE II environment.

On any new installation I execute MAKEFLIP.XXX one time and it generates FLIP.IT for whatever machine it is on. Now, whenever I need to change the command file default extension I type .DO FLIP.IT.

MAKEFLIP.XXX is not only a very useful utility, but you should get some good insight into how to write code that writes code. This could be especially helpful in custom installations.

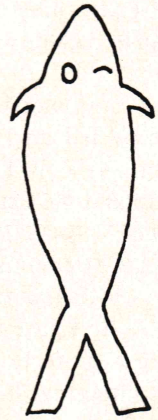
Before I get calls and letters telling me that my search routine is too loose and could find a bogus patch, let me say that I already know that. However, I have tried MAKEFLIP.XXX on several types of micros and most versions of dBASE II with complete success. The chances of finding an invalid patch location are too small to justify additional code. Besides, you will know right away if you found a bad patch area. *It won't work!* **DDJ**

How to write dBASE code that writes dBASE code.

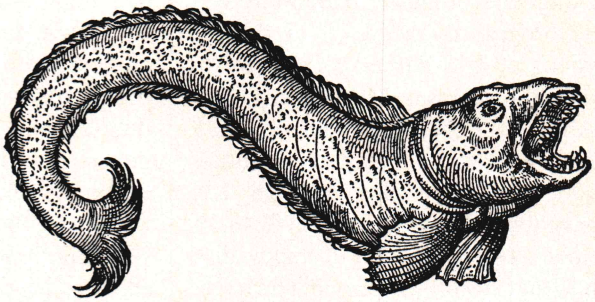
(Listing begins on next page)

Gene Head, Head Quarters, 2860 NW Skyline, Corvallis, OR 97330.

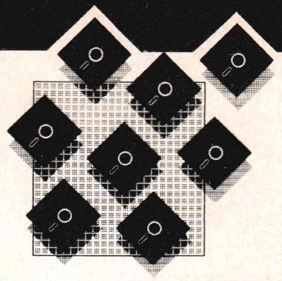
```
*
*   Another dBASE II goody from:
*
*           Head Quarters
*
*           Gene Head
*           2860 NW Skyline Drive
*           Corvallis, Oregon  97330
*           (503)  758-0279
*
*   For non-commercial use only.  If you use this utility
*   to help you develop any software that earns you money
*       (even in your own business) send me $5.00.
*   It's a fair price to pay and worth a clear conscience.
*
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*
*
*   Program name --> MAKEFLIP.XXX
*
*             (DO NOT change the name of this command file)
*             (Execute it as .DO MAKEFLIP.XXX or it may not work)
*
*   * * *  This program generates the command file FLIP.IT  * * *
*
*   You only need to run this program ONCE to create FLIP.IT.
*   After you create FLIP.IT put a copy on your dBASE II disk.
*   Then when you wish to change default command file extension
*   type .DO FLIP.IT
*   Example:
*
*   To run CP/M command files defaulted to .CMD under MS-DOS simply
*   type .DO FLIP.IT.  To change back to the .PRG default, type the
*   same thing again, .DO FLIP.IT.
*
*   Likewise, to run MS-DOS command files defaulted to .PRG under
*   CP/M, type .DO FLIP.IT.  To change back to .CMD default, again
*   type .DO FLIP.IT.
*
*


SET TALK OFF

*  This range works well for dBASE II versions 2.4,
*  2.41 and Z-80 2.41.  For other versions they may have to change.
STORE '12300' TO LOW
STORE '12500' TO HIGH


*  First get the range of program RAM to search

*     for the default extension

STORE 'Y' TO CHOICE
DO WHILE CHOICE = 'Y'

    STORE T TO RANGE

    ERASE

    DO WHILE RANGE
      @ 12,10 SAY 'Enter START of search area.  Suggest --> ';

      GET LOW PICTURE '99999'

      @ 14,10 SAY '  Enter END of search area.  Suggest --> ';

      GET HIGH PICTURE '99999'

      READ


      STORE VAL(LOW) TO MLOW
      STORE VAL(HIGH) TO MHIGH

        IF MLOW >= MHIGH .OR. MHIGH > 65000 .OR. MLOW < 100
          @ 20,10 SAY 'SEARCH AREA OUT OF RANGE.  TRY AGAIN . . .'
        ELSE

          STORE F TO RANGE
        ENDIF MLOW >= MHIGH .OR. MHIGH > 65000 .OR. MLOW < 100
```



```
ENDDO WHILE RANGE

ERASE

*  This is not the most accurate testing but the chances are
*    very high that it will only find what we are looking for.

@ 12,10 SAY 'SEARCHING'
STORE F TO FOUND

DO WHILE (.NOT. FOUND) .AND. MLOW < MHIGH

     IF PEEK(MLOW) <> 67 .AND. PEEK(MLOW) <> 80
      STORE MLOW+1 TO MLOW
      LOOP
     ELSE
       IF PEEK(MLOW+1) <> 77 .AND. PEEK(MLOW) <> 82
     STORE MLOW+1 TO MLOW
     LOOP
       ELSE
         IF PEEK(MLOW+2) <> 68 .AND. PEEK(MLOW) <> 71
             STORE MLOW+1 TO MLOW
             LOOP
             ELSE
             STORE T TO FOUND
             LOOP
         ENDIF PEEK(MLOW+2) <> 68 .AND. PEEK(MLOW) <> 71
       ENDIF PEEK(MLOW+1) <> 77 .AND. PEEK(MLOW) <> 82
     ENDIF PEEK(MLOW) <> 67 .AND. PEEK(MLOW) <> 80

ENDDO WHILE (.NOT. FOUND) .AND. MLOW < MHIGH

* If we found our patch area create a command file called FLIP.IT
IF FOUND
@ 12,10 SAY 'CREATING COMMAND FILE --> FLIP.IT '
STORE STR(MLOW,5) TO PATCH
SET ALTERNATE TO FLIP.IT
SET ALTERNATE ON
SET CONSOLE OFF
? [*  FLIP.IT]
? [*]
? [*  FLIP.IT WILL FLIP COMMAND FILE EXTENSION  .CMD <--> .PRG]
? [*]
? [*  FLIP.IT IS DESIGNED FOR THOSE WHO DEVELOP PROGRAMS USING CP/M]
? [*  dBASE II TO RUN UNDER MS-DOS dBASE II AND GET TIRED CHANGING]
? [*  THE COMMAND FILE EXTENSIONS FROM .CMD TO .PRG AND VICE-VERSA!]
? [*]
? [*  FLIP.IT CAUSES YOUR CP/M DBASE II TO ACCEPT .PRG AS DEFAULT]
? [*  COMMAND FILE EXTENSION.  IT CAN ALSO LET YOUR MS-DOS dBASE II]
? [*  ACCEPT .CMD AS THE DEFAULT COMMAND FILE EXTENSION.]
? [*]
? [*  IN EITHER CASE JUST TYPE .DO FLIP.IT]
? [*]
?
? [IF PEEK(]+PATCH+[) = 67]
? [ POKE ]+PATCH+[, 80, 82, 71]
? [ ? 'DEFAULT COMMAND EXTENSION --> .PRG']
? [ELSE]
? [     IF PEEK(]+PATCH+[) = 80]
? [        POKE ]+PATCH+[, 67, 77, 68]
? [        ? 'DEFAULT COMMAND EXTENSION --> .CMD']
? [     ELSE]
? [        ? 'ERROR CHANGING DEFAULT EXTENSION']
           ? [ENDIF]
           ?
           SET ALTERNATE OFF
           SET ALTERNATE TO
           SET CONSOLE ON

           STORE 'N' TO CHOICE

           ELSE

           @ 12,10 SAY 'CAN NOT LOCATE PATCH AREA WITHIN;
        THE SPECIFIED RANGE.'
           @ 14,10 SAY 'INCREASE THE SEARCH RANGE.;
         TRY AGAIN? (Y/N) --> ' GET CHOICE
           READ

           ENDIF  FOUND

           ENDDO while CHOICE = 'Y'


           ERASE

               * * * * * * * * * * * * * * * * * * * * * *
               *  end of source code for MAKEFLIP.XXX   *
               * * * * * * * * * * * * * * * * * * * * * *
```



(End Listing)



Modula-2/86, Version 1.04

Company: Logitech, 805 Veterans Blvd., Redwood City, CA 94086
Operating System: MSDOS and CP/M 86
Price: $495.00 *(Discounts are available for educational institutions and user groups. — Ed.)*
Circle Reader Service No. 135
Reviewed by Michael Schmidt

There has been much to-do about Modula-2 recently, with the theme that Modula-2 is the successor to Pascal. But how does a new Modula-2 compiler compare with an extended Pascal compiler that's been around a while? I was curious to find out.

I pitted the Logitech Modula-2/86 compiler, version 1.04 (not a public release), against the Microsoft MS-Pascal compiler, version 3.13. MS-Pascal is a highly extended version of the Pascal language, designed to facilitate systems programming. Many of the extensions of MS-Pascal correspond to standard features of Modula-2.

All tests were performed on an 8 MHz 8086 Seattle Computer Products S-100 system, running MSDOS 2.0. The system was equipped with 224K of memory, two Qume DT-8 floppy disk drives that store 1.2Mb each, and a Zenith Z19 terminal. The system was not equipped with an 8087 coprocessor.

Documentation

The Logitech manual represents what I would call a "user's guide," with certain parts of a "reference manual" in numerous appendices. With this organization, two-thirds of the documentation is in appendices! The preface refers the reader to the book *Programming In Modula-2* by N. Wirth for a definition of the Modula-2 language. However, implementation specifics and peculiarities of the SYSTEM and library modules in the Logitech implementation are not described, of course, in Wirth's book, and Logitech reduces them to the lowly status of appendices.

What do I have against appendices? The problem lies not with appendices *per se* but with the authors, who use them here as an excuse to supply substandard documentation. Gone are any requirements that material be presented in a logical order or that coherent English appear on each page or that a 70-page section show subdivisions with page numbers in the table of contents.

The largest appendix consists of the definition modules of the Logitech library, with comments. Unfortunately, these comments are not always enough! For example, the rather complex library module FileSystem seems to defy comprehension, at least mine. A similar file system appears as one of two alternate designs (one for RT-11, one for Lilith) in Wirth's book but with little supporting text—again just a definition module. It should be obvious to anyone struggling through Modula-2 definition modules that they are not self-documenting and require a little English explanation.

Part of the difficulty I had understanding these library modules resulted from undefined terminology borrowed from an alien (non-MSDOS) environment. For instance, what the heck is a "medium"? And how am I supposed to know that I should precede a filename with "DK:" to select the default drive? These peculiarities are bad enough by themselves, but they become downright unmanageable when they are not documented.

The "user's guide" portions of the Logitech documentation are well written and complete, with the following exceptions. A discussion of the runtime support (RTS) was missing alltogether. This program resides in the file M2.EXE, and handles program loading, overlays, coroutine transfers, and runtime errors. It produces a number of runtime error messages, which are not listed anywhere in the manual. Also, the section on linking overlays was not sufficiently descriptive.

The MS-Pascal documentation is more hefty. It consists of both a user's manual and a separate, much larger, language reference manual. All the information I have ever needed to know about MS-Pascal is there somewhere! The mere bulk of these manuals would tend to overwhelm the first-time user, but actually, the organization is logical, and once you have read through the manuals a few times, you can retrieve information quickly.

Neither set of manuals serves as an introduction to the language that its compiler implements. However, the MS-Pascal documentation is logically complete by itself, whereas the Logitech documentation requires *Programming In Modula-2* for a formal definition of Modula-2. The saving grace of the Logitech documentation is Logitech's technical support. Whenever I reached an impasse, a quick phone call on the Logitech 'hotline' produced my answer. With Microsoft, I am not even aware of a number I can call.

Installation

The MSDOS version of the Logitech compiler is distributed only on three 5¼-inch double-sided floppies, using 9-sector/track formatting. Logitech was not able to satisfy my request for an 8-inch distribution disk. I was able to add a minifloppy drive to my system (most MSDOS systems use the minifloppies anyhow), but for a minority of MSDOS systems this will present a problem.

The Logitech manual instructs the user to place the command

device = ANSI

in the CONFIG.SYS file. This is re-

quired only for systems with non-ANSI console devices. Because my Z19 is set to ANSI mode, no special driver is needed. Only the Logitech debugger generates ANSI escape sequences; you can operate both the compiler and linker without an ANSI console device.

The rest of the installation depends on whether you are using floppies or a Winchester. Actually, all that matters is that you have a mass-storage device that can hold about 1Mb, which is what a Modula-2 system disk requires. My 8-inch floppies were adequate. If you have a system with only 360K 5¼-inch floppies, you will need three diskettes in addition to the one holding your source code: a system disk with an editor on it, a compiler disk, and a linker/debugger disk. Good luck!

If you have adequate mass storage, you are instructed to place the following commands in your AUTOEXEC.BAT file:

```
SET M2LIB=\M2LIB\SYM
SET M2LNK=\M2LIB\LNK
SET M2REF=\M2LIB\REF
SET M2MAP=\M2LIB\MAP
```

I preceded each of the above symbols with a drive designator (M2LIB = A:\M2LIB\SYM) so that I did not have to set my default drive to the Modula system disk. The files on the distribution disks are copied to the above subdirectories or to the subdirectory \M2LOD for the executable files. Unfortunately, no corresponding symbol for this latter subdirectory exists (the command SET M2LOD=A:\M2LOD does nothing), and if the default drive does not contain the Modula system disk, the complete path name of the executable file must be given. In addition, the RTS in the file M2.EXE must be copied into a subdirectory selected by the MSDOS PATH command (in my directory structure this was A:\BIN).

Two example programs are provided on the distribution disk. When I first tried to compile them, I received an error message "cannot load. . ." during linking. What disturbed me was that this message was not in the manual. I suspected a lack of RAM and on closer examination discerned that the manual requires 192K of *program* RAM, which does not include the roughly 32K required by MSDOS 2.0. I upgraded my system to 256K and had no further troubles. To avoid such confusion, Logitech should admit to a requirement of 256K of system RAM.

The Logitech distribution is configured for an IBM PC. The source code for several modules, which might need modification for a different environment, the object files for the compiler, linker, and debugger themselves, and the assembler source code of the RTS modules are standard distribution! A rather cryptic comment in an appendix states that the user should study the source code for these and make changes as required.

Despite my different hardware configuration, I found that none of the library modules had to be recompiled. However, the process/interrupt module of the RTS proved to be incompatible with my system. It assumes a single 8259A interrupt controller, addressed at I/O ports 20 and 21H, as is the case in the IBM PC. My system has multiple cascaded 8259As and nothing at I/O ports 20 and 21H! This module plays with the interrupt controller for the TRANSFER and IOTRANSFER operations to implement an obscure feature of Modula-2 that allows numeric "priority" module parameters to turn modules into "monitors," as in Concurrent Pascal. Logitech interprets module priorities as corresponding to different interrupt masks for a single interrupt controller. Luckily, the compiler, linker, and debugger do not seem to use prioritized modules or TRANSFER statements and thus can be run without modification of M2.EXE. My system would require nontrivial modification of this assembler module to get these Modula-2 features working.

The installation of the MS-Pascal compiler is more straightforward. All that is required is to copy the three passes of the compiler, one of the library files, and the special "LARGE" version of MS-LINK to a system disk. These files can be combined on a single Pascal system disk complete with an editor—even a 360K 5¼-inch floppy. The MS-Pascal compiler (and linker!) require less than 160K total system RAM to operate.

Compiling and Linking

Definition and implementation modules are compiled separately by the Logitech compiler. The output from compiling a definition module is accessed automatically when compiling an implementation module. You can invoke all four passes of the Logitech compiler entirely from the command line with the command:

```
M2 COMP <filename>
```

The overlay mechanism of the RTS loads each pass of the compiler into memory separately. For this to work, however, the default drive when the compiler is invoked must be the drive that contains the directory \M2LOD. To get around this and to automatically invoke some of the compiler switches, I created the following command procedure (MOD.BAT) to invoke the compiler:

```
M2 A:\M2LOD\COMP
    %1/R-/S-/T-/E
```

Similarly, I created a command procedure MLNK.BAT to invoke the linker:

```
M2 A:\M2LOD\LINK %1
```

The compiler and linker use the first eight characters of a module name to search for compiled definition modules and object modules. This has the advantage of automating the search for library modules.

As is apparent from the compile and link times shown in the Figure (page 76), the Logitech compiler and linker are slow. A further annoyance stems from the Logitech compiler's organization into four passes, the first three of which can generate syntax-related error messages. Whether the compilation is aborted on an error after either of the first two passes or not can be configured by the user. This situation is made worse by the compiler's lack of certain types of error recovery. For instance, the compiler is case sensitive: all keywords are in upper case. It is very easy to forget and write "to" instead of "TO" in a FOR loop. Even though it should be an easy matter for the compiler to flag this as a warning and continue, such errors are treated as fatal.

The MS-Pascal compiler can also be invoked entirely from the command line. However, each pass of the compiler must be invoked separately; no command procedure is provided which automates this process. I have written such a command procedure (PAS.BAT) to invoke the required passes of the compiler:

```
PAS1 %1,%1,,%1;
IF ERRORLEVEL 0 DEL %1.LST
```

```
IF ERRORLEVEL 4
    GOTO FINISHED
PAS2
:FINISHED
```

This command procedure would need enhancement to use PAS3, which provides a pseudo-assembly language listing of the code generated. In order to link multiple modules using MicroSoft's LINK utility, the following command format offers the greatest flexibility (the /NO parameter suppresses the automatic search for the Pascal library on the default path):

```
LINK/NO @program
```

A "program" file must be created for each program you are developing. It would contain the following entries:

```
program+
module1+
module2+
moduleN
program
program
A:\LIB\PASCAL
```

Having to maintain this additional file can sometimes be a nuisance.

Although the MS-Pascal compiler is not blindingly fast, it is certainly tolerable—over twice as fast as the Logitech compiler. The MS-Pascal compiler is organized as three passes, only the first of which generates syntax-related error messages. The error recovery of the compiler is adequate, and some common syntax errors are automatically corrected, generating only warnings. However, some simple types of errors, such as a missing END, will cascade and cause an avalanche of meaningless error messages.

The MS-Pascal compilation unit that most closely resembles the Modula-2 definition and implementation modules is the "unit" concept borrowed from UCSD Pascal. A unit also consists of a definition part and an implementation part. However, the definition portion of an MS-Pascal unit is not itself compiled; it is included as part of the source of its implementation, as well as any compilation units importing it, usually by means of the INCLUDE directive. This allows for "version" errors, resulting in changes in such definition parts.

Benchmarks

I decided to use the system clock to time my benchmarks. Although both Logitech Modula-2 and MS-Pascal provide library functions, I didn't care for the format of either. Instead I decided to write a GetTime procedure that would perform an MSDOS function call. I immediately ran into problems with MS-Pascal.

MS-Pascal provides a library function (DOSXQQ) to invoke MSDOS functions and two global variables (CRCXQQ and CRDXQQ) to pass parameters through the CX and DX registers. This function does not work properly: the two global variables are not updated from the CX and DX registers following a MSDOS call. Furthermore, even if this mechanism worked as advertised, it still would not be powerful enough to handle all MSDOS functions. The registers accessed by various MSDOS functions include the AX, BX, CX, DX, SI, DI, DS, and ES registers! I wrote a general purpose MSDOS interface procedure for MS-Pascal in assembly language (Listing One, page 78) which allows access to all these registers, and seems to do the job quite nicely. Logitech, on the other hand, supplies a much more versatile library function, DOSCALL, which gives full access to all MSDOS functions and worked properly on all my test cases.

I wrote a general purpose clock module in both Modula-2 and MS-Pascal (Listing Two, page 80, and Listing Three, page 82). It reflects the capabilities of MSDOS by providing a function GetTime, which returns the time through a record. I then wrote a simple stopwatch module (Listing Four, page 84, and Listing Five, page 86) which provides two procedures, Start and Stop. This module allowed me to measure the benchmarks to within a hundredth of a second.

I ran three benchmarks: the ever-popular Sieve of Eratosthenes (Listing Six, page 86 and Listing Seven, page 88), the Fibonacci number benchmark (Listing Eight, page 88, and Listing Nine, page 90), and a skeletal text filter program (Listing Ten, page 92, and Listing Eleven, page 93) that does nothing more than copy files. Both of these compilers produced very fast, highly optimized code (see the Figure). The code generator of the MS-Pascal compiler performed better on the Sieve benchmark, but not by a

benchmark & compiler	compile time (sec)	link time (sec)	code size (bytes)	linked size (bytes)	execute time (sec)
1. Sieve of Eratosthenes					
Logitech Modula-2/86	82	69	243	21474	5.25
Microsoft MS-Pascal	26	38	161	19518	4.87
2. Fibonacci Numbers					
Logitech Modula-2/86	81	68	141	21362	17.67
Microsoft MS-Pascal	25	37	107	19470	23.18
3. Text Filter					
Logitech Modula-2/86	87	69	391	21694	7.83
Microsoft MS-Pascal	27	38	353	19742	18.32

Figure

Benchmark Results.
All tests performed on 8MHz 8086, all run-time error-checking disabled.

wide margin. The Logitech compiler won the other two benchmarks by a somewhat wider margin. However, both of these languages produced bulky executable files by the time the linkers were through with them.

One thing that is not apparent from the benchmarks is that the Logitech linker produces very small executable files when none or only a few of the library modules are imported. The library module InOut (which most programs access) causes large executable files because it pulls in the FileSystem module; if no file support is needed, the Terminal module produces small executable files. This feature would be of value for writing certain types of utility programs, such as a program to continuously display the time.

Miscellaneous

Logitech supplies a *post mortem* symbolic debugger with the compiler package. This program allows the user to examine the procedure-calling sequence, last statement executed, and value of all data items following a 'core dump.' Such a core dump is performed automatically by the RTS following a runtime error or a ctrl-C typed by the user during execution. This provides significantly more diagnostic information than the simple error message produced by MS-Pascal on a runtime error. However, it is not as powerful as an interactive runtime debugger; such a product has been announced by Logitech, however.

Both Logitech and MicroSoft should be commended for the large number of useful library modules supplied with their compilers. Despite quirks in both libraries, these give the programmer a tremendous headstart in developing an application program.

Both of these compilers provide for floating-point support, either using an 8087 coprocessor or with an 8087 emulation library. I did not perform a benchmark on the emulation library, because anyone who is truly interested in fast floating point would be using an 8087, anyway.

The Logitech compiler does not support a 32-bit integer type, as does MS-Pascal. This is unfortunate because a 16-bit integer is too small for numerous applications (random number generators, for instance). This problem will undoubtedly be resolved in a future release, as recent revisions to the Modula-2 language by N. Wirth include LONGCARD and LONGINT data types.

My final complaint about the Logitech compiler, which I have already touched on, is that when a ctrl-C is typed during compilation or linking, about 15 seconds elapse and a large MEMORY.PMD file is created for the debugger in the default directory. This can be changed only by reassembling the RTS after changing one of the parameters in the source code. Of course, *post mortem* debugging of application programs requires a version the RTS with this feature left intact. I would suggest that Logitech distribute two executable versions of the RTS.

Conclusion

The Logitech Modula-2/86 package is clearly a winner. It provides a complete, relatively mature Modula-2 environment for the 8086 that is competitive with the best 8086 compilers available. In fact, a comparison of its speed on my system with published benchmarks on the Lilith shows them to be almost even in execution time. Perhaps most importantly, Logitech provides the kind of technical support necessary for a professional software development package.

On the other hand, MS-Pascal is still very much a viable systems language. It would certainly be preferable on a small system: it requires 96K less RAM and can perform adequately on a system equipped only with 5¼-inch floppies. In addition, its faster compile and link times would be appreciated on the much slower IBM PC.

I suppose the acid test is this: Which of these compilers will I be using from now on? This brings us back to the beginning of the article. Modula-2 really is the successor to Pascal, and the Logitech compiler is a thoroughly usable implementation. I have made the switch!

DDJ

(Listings begins on next page)

Reviews (Text begins on page 74)

Listing One

```
INTERFACE; UNIT SYSTEM (
  Registers,
  DOSCALL);

TYPE

  Registers = RECORD
    ax, bx, cx, dx: WORD;
    si, di: WORD;
    ds, es: WORD;
  END;

PROCEDURE DOSCALL (VAR r: Registers);
{
    loads 8086 registers from r
    invokes MS-DOS
    loads r from 8086 registers
}

END; {SYSTEM}
        NAME     SYSTEM

        PUBLIC   DOSCALL

RGSTRS  STRUC
rax     DW       ?
rbx     DW       ?
rcx     DW       ?
rdx     DW       ?
rsi     DW       ?
rdi     DW       ?
rds     DW       ?
res     DW       ?
RGSTRS  ENDS

DOSCODE SEGMENT

        ASSUME   CS: DOSCODE

DOSCALL PROC     FAR
        PUSH     BP                  ; save calling frame pointer
        MOV      BP, SP              ; get doscall frame pointer
        PUSH     DS                  ; save calling data segment
        MOV      BP, 6[BP]           ; get rgstrs pointer

; load 8086 registers from structure
        MOV      AX, [BP].rax
        MOV      BX, [BP].rbx
        MOV      CX, [BP].rcx
        MOV      DX, [BP].rdx
        MOV      SI, [BP].rsi
        MOV      DI, [BP].rdi
        MOV      DS, [BP].rds
        MOV      ES, [BP].res
```

(Continued on page 80)

QUALITY SOFTWARE AT REASONABLE PRICES

CP/M Software by
Poor Person Software

Poor Person's Spooler **$49.95**
All the function of a hardware print buffer at a fraction of the cost. Keyboard control. Spools and prints simultaneously.

Poor Person's Spread Sheet **$29.95**
Flexible screen formats and BASIC-like language. Pre-programmed applications include Real Estate Evaluation.

Poor Person's Spelling Checker **$29.95**
Simple and fast! 33,000 word dictionary. Checks any CP/M text file.

aMAZEing Game **$29.95**
Arcade action for CP/M! Evade goblins and collect treasure.

Crossword Game **$39.95**
Teach spelling and build vocabulary. Fun and challenging.

Mailing Label Printer **$29.95**
Select and print labels in many formats.

Window System **$29.95**
Application control of independent virtual screens.

All products require 56k CP/M 2.2 and are available on 8" IBM and 5" Northstar formats, other 5" formats add $5 handling charge. California residents include sales tax.

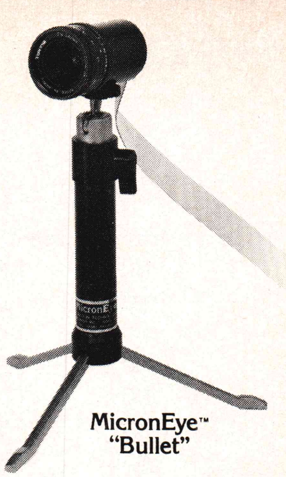
Poor Person Software
3721 Starr King Circle
Palo Alto, CA 94306
tel 415-493-3735

CP/M is a registered trademark of Digital Research

Circle **no. 71** on reader service card.

Now Your Computer Can See!

$295.00*



MicronEye™ "Bullet"

A total imaging system complete and ready for plug-and-go operation with your personal computer.

The MicronEye™ offers selectable resolution modes of 256 x 128 and 128 x 64 with operating speeds up to 15 FPS. An electronic shutter is easily controlled by software or manual functions, and the included sample programs allow you to continuously scan, freeze frame, frame store, frame compare, print and produce pictures in shades of grey from the moment you begin operation.

Only the MicronEye™ uses the revolutionary IS32 OpticRAM™ image sensor for automatic solid state image digitizing, with capability for grey-tone imaging through multiple scans. And with these features, the MicronEye™ is perfectly suited for graphics input, robotics, text and pattern recognition, security, digitizing, automated process control and many other applications.

The MicronEye™ is available with immediate delivery for these computers: Apple II, IBM PC, Commodore 64 and the TRS-80CC (trademarks of Apple Computer Inc., International Business Machines, Commodore Corp., and Tandy Corp. respectively).

Phone for MicronEye™ information on the Macintosh, TI PC and RS232 (trademarks of Apple Computer Inc. and Texas Instruments respectively.)

*(Add $10.00 for shipping and handling [Federal Express Standard Air]; residents of the following states *must add sales tax:* AK, AZ, CA, CO, CT, FL, GA, IA, ID, IL, IN, LA, MA, MD, ME, MI, MN, NC, NE, NJ, NY, OH, PA, SC, TN, TX, UT, VA, VT, WA, WI.)

MICRON TECHNOLOGY, INC.

SYSTEMS GROUP
1475 Tyrell Lane
Boise, Idaho 83706
(208) 386-3800
TWX 910-970-5973

Circle **no. 96** on reader service card.

THE LEGENDARY WAY TO SOLVE YOUR BACKUP PROBLEMS

Here's what Microsystems had to say about our original product: "QBAX will probably become one of those legendary programs that everyone eventually buys. It performs a function useful to anyone with a CP/M system, does it well and quickly, is understandable to the novice computer user."

*"Every time you run QBAX, the program determines which of your disk files has been changed since the last time it was run. Then it copies these files, and **only** these files, to whatever disk you specify. This is called **incremental backup,** and is the backup method of choice on most large timesharing systems. It will work on any or all active user areas, and so is an absolute **must** for hard- or RAM-disk owners."*

©1983 by Ziff-Davis Publishing Company

Announcing a major enhancement, Qbax2, specifically designed for hard disk users: ■ *incremental backup by extent* ■ *splits files larger than one floppy* ■ *smart restore: knows exactly which floppies to mount. Can restore individual files or wildcards* ■ *volume space recovery: prevents consuming floppies endlessly when the same files are backed up repeatedly.* ■ *time & date stamp & version number on all backup records.*

Qbax2 is $95. For floppy disk users Qbax1 is still available at $40.

Amanuensis, Inc.
R.D. #1 Box 236
Grindstone, PA 15442
(412) 785-2806

For CP/M 2.2 on 8" SSSD & popular 5¼" formats
MC, Visa accepted
OEM inquiries invited

Qbax TM Amanuensis, Inc.
CP/M Registered TM Digital Research

Shipping:
$2 U.S. & Canada, $4 overseas.

Circle **no. 5** on reader service card.

Listing One

```
        INT     21H                 ; call MS-DOS

; load structure from 8086 registers
        MOV     [BP].rax, AX
        MOV     [BP].rbx, BX
        MOV     [BP].rcx, CX
        MOV     [BP].rdx, DX
        MOV     [BP].rsi, SI
        MOV     [BP].rdi, DI
        MOV     [BP].rds, DS
        MOV     [BP].res, ES

        POP     DS                  ; restore calling data segment
        POP     BP                  ; restore calling frame pointer
        RET     2                   ; pop structure pointer during return

DOSCALL ENDP

DOSCODE ENDS

        END
```

End Listing One

Listing Two

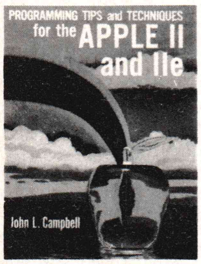
```
DEFINITION MODULE Clck;

EXPORT QUALIFIED
  Time,
  Date,
  SetTime,
  SetDate,
  GetTime,
  GetDate;

TYPE
  Time = RECORD
    hour: [0..23];
    minute: [0..59];
    second: [0..59];
    hundredth: [0..99];
  END;

  Date = RECORD
    year: [1980..2099];
    month: [1..12];
    day: [1..31];
  END;

PROCEDURE SetTime (VAR t: Time);

PROCEDURE SetDate (VAR d: Date);

PROCEDURE GetTime (VAR t: Time);

PROCEDURE GetDate (VAR d: Date);

END Clck.

IMPLEMENTATION MODULE Clck;

FROM SYSTEM IMPORT
  DOSCALL;

(***********************************)

PROCEDURE SetTime (VAR t: Time);

VAR
  al: [0..255];
  cx, dx: CARDINAL;

BEGIN
  WITH t DO
    cx := 256*hour + minute;
    dx := 256*second + hundredth;
    DOSCALL (2DH, cx, dx, al);
  END;
END SetTime;

(***********************************)

PROCEDURE SetDate (VAR d: Date);

VAR
  al: [0..255];
  cx, dx: CARDINAL;
```

(Continued on page 82)

ADVANCE YOURSELF AT SUBSTANTIAL SAVINGS

Up to 20% off!

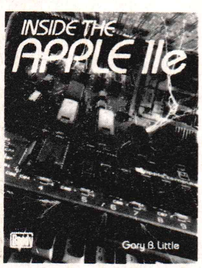
APPLE OWNERS:
Choose from—



Programming Tips and Techniques Kit for the Apple II/IIe
by L. L. Campbell

Takes an advanced look at the skills and techniques needed to solve common programming problems. Including handling input, processing, output and error entry. Offers supporting code to ease program writing and problem solving, and tips on using sound and hi/lo resolution graphics. All 38 programs from the book are on the accompanying diskette. Debugged, error-free and ready to run when you are.

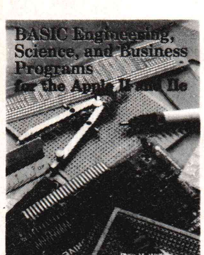
1984/403pp/paper & 1 diskette/D7661-2/$49.95



Inside the Apple IIe: Programming Access Tools Kit
by G. Little

Get an inside look at the inner workings of the 6502 microprocessor, the DOS 3.3 and PRO DOS operating systems—with explanations of how they really work, ROM, with a complete glossary of schematic drawings. This unique book/diskette tutorial has 30 programs (pre-keyed and ready to go) covering auxiliary memory, discussing and supplementing keyboard input routines, using double width graphics and more.

1984/310pp/paper & 1 diskette/D5556-8/$49.95

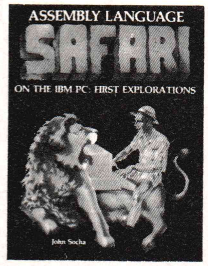


BASIC Engineering, Science and Business Programs for the Apple II/IIe Kit
by P. M. Wolfe and C. P. Koelling

From data manipulation and specific problem solving methods to forecasting and project scheduling, this "tool box" book/diskette combo offers 38 ready to run programs and examples. You'll get fast access to linear programming and regression, next event simulation, project planning and scheduling, forecasting with exponential smoothing, and more.

1985/320pp/paper & 1 diskette/D2905-0/$42.95

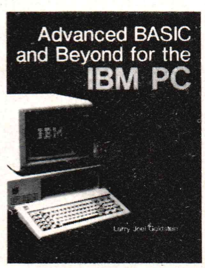
IBM PC OWNERS:
Choose from—



Software Tools for Assembly Language Safari on the IBM PC: First Explorations
by J. Socha

Informing without overwhelming, this new book/diskette package is a practical introduction to the inner workings of the 8088 microprocessor; machine language programming; and a sneak preview of more advanced features. The companion software has all the major programs from the book. Including *DSKPATCH*—an all inclusive program using the step-wise refinement approach to writing. And an *Advanced Version of DSKPATCH* (found only on the diskette) that allows you to do even more!

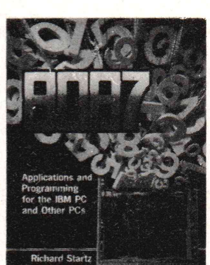
1984/256pp/paper & 1 diskette/D2948-0/$56.95



Advanced BASIC and Beyond for the IBM PC Programming Kit
L. J. Goldstein

Here's an indepth book/software tutorial to advanced BASIC programming skills. Covers files, graphis, event trapping, machine language, and subroutines. Includes a total of 60 prekeyed programs, such as a professional input routine, bar chart generator with spread sheet, character plotting routine, graphic screen dump routine, and more.

1984/368pp/paper & 1 diskette/D3251-8/$49.95



8087 Applications and Programming Kit for the IBM PC and Other PCs
by R. Startz

At last A *clear, complete* explanation of the number counting 8087 microprocesor for program writers and users. Starting comfortable and easy, the book/diskette package evolves into more detailed and technical language. The companion software gives you the source and object code for the book's 30 assembly language programs. Plus you'll be able to select programs in your choice of formats. And that selection includes BLOAD, so you can load and go right into BASIC.

1983/276pp/paper & 1 diskette/D4258-2/$54.95

You Decide How Much You Save—

*Buy 1—Get 5% off
*Buy 2—Get 10% off
*Buy 3 or more and you save a full 20%

Your Satisfaction Guaranteed Or Your Money Back!

If, within 15 days, you are not satisfied with your order, send it back for a complete refund. No questions asked.

Use the attached postpaid order card. Or the coupon below. Send to: Brady Communications Co., Inc.; Dept DB, Bowie, MD 20715

YES! I want to advance substantially! Here's how much I want to save:

☐5%
☐10% ☐20%

Send me:

☐ **Programming Tips and Techniques Kit for the Apple II/IIe** D7661-2/$49.95
☐ **Inside the Apple IIe: Programming Access Tools Kit** D5556-8/$49.95
☐ **BASIC Engineering, Science and Business Programs for the Apple II/IIe** D2905-0/$42.95
☐ **Software Tools for Assembly Language Safari on the IBM PC: First Explorations** D2948-2/$54.95
☐ **Advanced BASIC and Beyond for the IBM PC Programming Kit** D3251-8/$49.95
☐ **8087 Applications and Programming Kit for the IBM PC and Other PCs** D4258-2/$56.95

☐ Here's my check/mo including local sales tax
☐ Here's my
____ VISA ____ MasterCard
________________Number/Exp. Date
________________Signature

Name ________________
Address ________________
City ________________
State/Zip ________________

Dept. Y Y0530-DD(7)

BRADY

BRADY COMMUNICATIONS COMPANY INC.
A Prentice-Hall Publishing Company
BOWIE, MARYLAND 20715

Circle **no. 21** on reader service card.

Listing Two

```
BEGIN
  WITH d DO
    cx := year;
    dx := 256*month + day;
    DOSCALL (2BH, cx, dx, al);
  END;
END SetDate;

(***********************************)

PROCEDURE GetTime (VAR t: Time);

VAR
  cx, dx: CARDINAL;

BEGIN
  WITH t DO
    DOSCALL (2CH, cx, dx);
    hour := cx DIV 256;
    minute := cx MOD 256;
    second := dx DIV 256;
    hundredth := dx MOD 256;
  END;
END GetTime;

(***********************************)

PROCEDURE GetDate (VAR d: Date);
VAR
  cx, dx: CARDINAL;

BEGIN
  WITH d DO
    DOSCALL (2AH, cx, dx);
    year := cx;
    month := dx DIV 256;
    day := dx MOD 256;
  END;
END GetDate;

END Clck.
```

End Listing Two

Listing Three

```
INTERFACE; UNIT CLOCK (
  Time,
  Date,
  SetTime,
  SetDate,
  GetTime,
  GetDate);

TYPE
  Time = RECORD
    hour: 0..23;
    minute: 0..59;
    second: 0..59;
    hundredth: 0..99;
  END;

  Date = RECORD
    year: 1980..2099;
    month: 1..12;
    day: 1..31;
  END;

PROCEDURE SetTime (VAR t: Time);

PROCEDURE SetDate (VAR d: Date);

PROCEDURE GetTime (VAR t: Time);

PROCEDURE GetDate (VAR d: Date);

END; {Clock}

IMPLEMENTATION OF CLOCK;

USES SYSTEM;

{***********************************}

PROCEDURE SetTime;

VAR
  r: Registers;

BEGIN
  WITH t DO WITH r DO BEGIN
    ax := #2D00;
    cx := BYWORD (hour, minute);
    dx := BYWORD (second, hundredth);
    DOSCALL (r);
  END;
END; {SetTime}

{***********************************}

PROCEDURE SetDate;

VAR
  r: Registers;

BEGIN
  WITH d DO WITH r DO BEGIN
    ax := #2B00;
    cx := year;
    dx := BYWORD (month, day);
    DOSCALL (r);
  END;
END; {SetDate}
```

(Continued on page 84)

Listing Three

```
{***********************************}

PROCEDURE GetTime;

VAR
  r: Registers;

BEGIN
  WITH t DO WITH r DO BEGIN
    ax := #2C00;
    DOSCALL (r);
    hour := HIBYTE (cx);
    minute := LOBYTE (cx);
    second := HIBYTE (dx);
    hundredth := LOBYTE (dx);
  END;
END; {GetTime}

{***********************************}

PROCEDURE GetDate;


 VAR
   r: Registers;

 BEGIN
   WITH d DO WITH r DO BEGIN
     ax := #2A00;
     DOSCALL (r);
     year := cx;
     month := HIBYTE (dx);
     day := LOBYTE (dx);
   END;
 END; {GetDate}

 END. {Clock}
```

End Listing Three

Listing Four

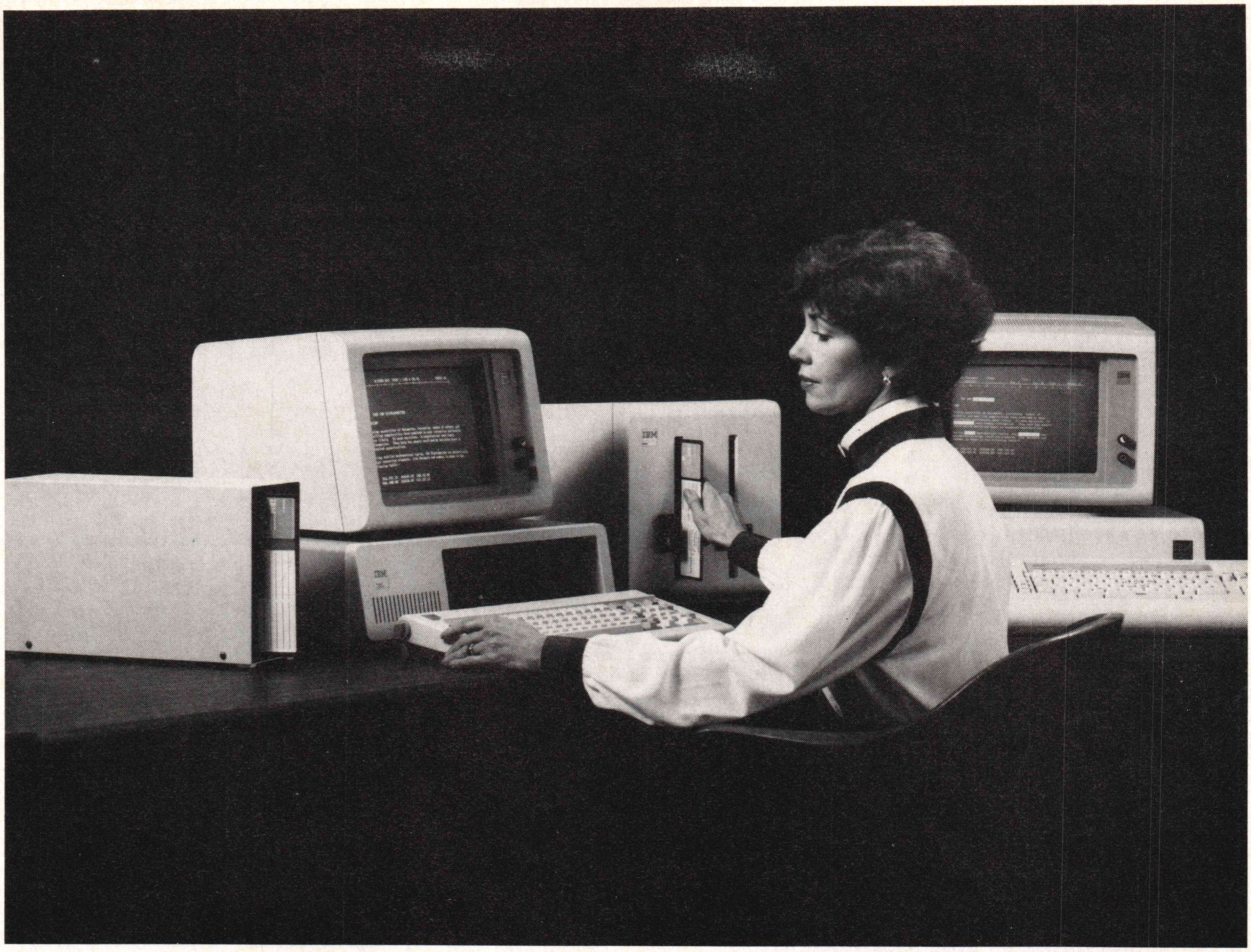
```
DEFINITION MODULE StopWatch;

EXPORT QUALIFIED
  Start,
  Stop;

PROCEDURE Start;
PROCEDURE Stop;

END StopWatch.
```

```
IMPLEMENTATION MODULE StopWatch;

FROM InOut IMPORT
  WriteString,
  WriteCard,
  WriteLn;

FROM Clck IMPORT
  Time,
  GetTime;

VAR
  t1, t2: Time;

(******************************)

PROCEDURE Start;

BEGIN
  GetTime (t1);
END Start;

(******************************)

PROCEDURE Stop;

(******************************)

PROCEDURE DisplayTime (t: Time);

BEGIN
  WITH t DO
    WriteCard(hour, 8);
    WriteCard(minute, 8);
    WriteCard(second, 8);
    WriteCard(hundredth, 8);
    WriteLn;
  END;
END DisplayTime;

(******************************)

BEGIN
  GetTime (t2);
  WriteString('start: '); DisplayTime(t1);
  WriteString('stop:  '); DisplayTime(t2);
END Stop;

END StopWatch.
```

End Listing Four

(Listing five begins on page 86)

Change your diskette to fit the IBM PC



THE FILE CONNECTION

8" DISKETTE SYSTEM FOR THE IBM PC

Our "FILE CONNECTION" programs provide 8" diskette file exchange between the IBM PC and most Micro-Mini-Main Frame computer systems.

Our "WORD CONNECTION" programs provide 8" diskette text document exchange between the IBM PC and many word processing systems.

Our "DISPLAYWRITER CONNECTION" programs transform documents from Textpack, Wordstar, Multimate, etc. to the DisplayWrite 2 format.

In addition to our hardware and program products, we also provide a conversion service for customer supplied diskettes. Please contact us for information about the hundreds of 5¼" and 8" diskette formats and systems which we currently support.

FLAGSTAFF ENGINEERING / P.O. Box 1970 / Flagstaff, AZ 86002
Telephone 602-774-5188 / Telex 705609 FLAG-ENG-UD

Circle **no. 25** on reader service card.

Listing Five

```
INTERFACE; UNIT StopWatch (
  Start,
  Stop);

PROCEDURE Start;
PROCEDURE Stop;

END; {StopWatch}

IMPLEMENTATION OF StopWatch;

USES Clock;

VAR
  t1, t2: Time;

{******************************}

PROCEDURE Start;

BEGIN
  GetTime (t1);
END; {Start}

{******************************}

PROCEDURE Stop;

{******************************}

PROCEDURE DisplayTime (t: time);

BEGIN
 WITH t DO BEGIN
   WriteLn(hour, minute,second,hundredth);
  END;
END; {DisplayTime}

{******************************}

BEGIN
  GetTime (t2);
  Write('start: '); DisplayTime(t1);
  Write('stop:  '); DisplayTime(t2);
END; {Stop}

END. {StopWatch}
```

End Listing Five

Listing Six

```
MODULE Benchmark1;
(* Sieve of Eratosthenes *)

FROM StopWatch IMPORT
  Start,
  Stop;

CONST
  iterations = 10;

VAR
  i: INTEGER;

(*********************************************)

PROCEDURE Sieve;

CONST
  size = 8190;

VAR
  i, prime, k, count: INTEGER;
  flags: ARRAY[0..size] OF BOOLEAN;

BEGIN
  count := 0;
  FOR i := 0 TO size DO
    flags[i] := TRUE;
  END;
  FOR i := 0 TO size DO
    IF flags[i] THEN
      prime := i + i + 3;
      k := i + prime;
      WHILE k <= size DO
        flags[k] := FALSE;
        INC (k, prime);
      END;
      INC (count);
    END;
  END;
END Sieve;

(*********************************************)

BEGIN
  Start;
  FOR i := 1 TO iterations DO
    Sieve;
  END;
  Stop;
END Benchmark1.
```

End Listing Six

(Listing seven begins on page 88)

A Proffessional Quality Z80/8080/8085 Disassembler

WHEN YOU NEED SOURCE FOR YOUR CODE

you need REVAS 3

REVAS interactively helps you:

- Analyse your software for modification
- disassemble files as large as 64K
- Assign Real labels in the disassembly
- Insert COMMENTS in the disassembly
- Generate a Cross Reference (XREF) listing

A 60 page manual shows how the powerful **REVAS** command set gives you instant control over I/O to files, printer, or console; how to do a disassembly; and even how the disassembler works! You get on line help, your choice of assembler mnemonics, control of data interpretation, and calculation in any number base!

REVAS runs in Z80 CPM computers; is available on 8" SSSD (standard), RAINBOW, and other (ask) formats

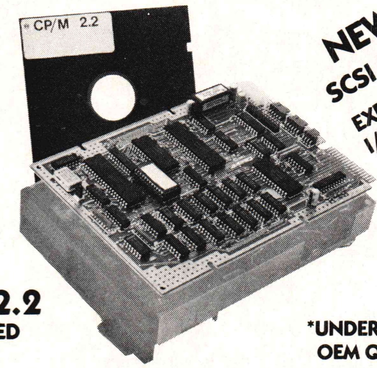
Price: $90.00 (plus applicable tax), Manual only: $15.00

REVASCO
6032 Chariton Ave., Los Angeles, CA 90056
Voice: (213) 649-3575 Modem: (213) 670-9465

Circle **no. 80** on reader service card.

The Little Board™...$349*

The world's simplest and least expensive CP/M computer



NEW SCSI/PLUS™ EXPANSION I/O OPTION

CP/M 2.2 INCLUDED

***UNDER $200 IN OEM QUANTITIES**

- **4 MHz Z80A CPU, 64K RAM, Z80A CTC, 2732 Boot ROM**
- **Mini/Micro Floppy controller (1-4 Drives, Single/Double Density, 1-2 sided, 40/80 track)**
- **Only 5.75 x 7.75 inches, mounts directly to a 5 1/4" floppy drive**
- **2 RS232C Serial Ports (75-9600 baud & 75-38,400 baud), 1 Centronics Printer Port**
- **Power Requirement: +5VDC at .75A; +12VDC at .05A/On-board -12V converter**
- **CP/M 2.2 BDOS • ZCPR3 CCP • Enhanced AMPRO BIOS**
- **AMPRO Utilities included:**
  - **read/write to more than 2 dozen other formats (Kaypro, Televideo, IBM CP/M86....)**
  - **format disks for more than a dozen other computers**
  - **menu-based system customization**
- **BIOS and Utilities Source Code Available**
- **SCSI/PLUS Adapter:**
  - **Mounts directly to Little Board • Slave I/O board control • Full ANSC X3T9.2**
  - **16 bidirectional I/O lines • $99/Quantity 1**

AMPRO COMPUTERS, INCORPORATED

Distributor/Dealer/Reps Inquiries Invited

Z80A is a registered trademark of Zilog, Inc.
CP/M is a registered trademark of Digital Research.

67 East Evelyn Ave. • Mountain View, CA 94041 • (415) 962-0230 • TELEX 4940302

Circle **no. 7** on reader service card.

ConIX™

UNIX™ Technology for CP/M™

ConIX can provide any 48K+ CP/M-80 compatible system with many advanced capabilities of UNIX. You'll be amazed at what your 8-bit micro can do now! ConIX features include:

I/O Redirection and Pipes (uses memory or disk), multiple commands per line, full upper/lower case and argument processing, Auto Screen Paging, Programmable Function Keys, improved User Area Directory manipulation, Command and Extension (Overlay) Path Searching, "Virtual" disk system, 8Mb Print Spooler, extensive preprocessed "Shell" command programming language, 300+ variables, over 100 built-in commands, Math Package, 22 new BDOS SysCalls, Archiver (compacts files for disk space savings of over 50%), On-Line Manual System, and much more! Uses as little as 1/2K RAM! Runs with CP/M for true data and software compatibility. Installs quickly and easily without any system modifications.

The ConIX Operating System
List Price: $165

Price includes Instructional Manual, 8" SSSD disk, and free support. 5¼" format conversions available. To order, contact your local dealer, or buy direct and add shipping: $4.50 UPS, $10 Canada, $25 overseas. COD $2 extra (USA only). NY State residents add sales tax.

Computer Helper Industries Inc.
P.O. Box 680 Parkchester Station, NY 10462
Tel. (212) 652-1786

Dealer inquiries invited!

UNIX: AT&T Bell Labs, CP/M: Digital Research, ConIX: Computer Helper Ind.

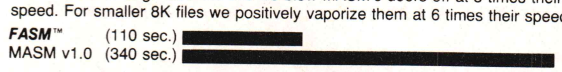
Circle **no. 22** on reader service card.

ASSEMBLE 3-6 TIMES FASTER ON THE IBM PC

Introducing ***FAST ASSEM-86***™, the first Editor/Assembler for the IBM PC and PC compatibles. ***FAST ASSEM-86***™ ***(FASM)*** is significantly faster and easier to use than the IBM Macro-Assembler (MASM). Whether you are new to assembly language and want to quickly write a small assembly language routine, or are an experienced MASM user tired of waiting months to assemble large files, ***FAST ASSEM-86*** will bring the excitement back to assembly language.

FAST ASSEM-86 **IS MUCH FASTER:**

- How fast is ***FASM***™? The graph below shows relative assembly times for a 48K source file. For large files like this we blow MASM's doors off at 3 times their speed. For smaller 8K files we positively vaporize them at 6 times their speed.



- ***FAST ASSEM-86*** is faster for the following reasons: **(1)** Written entirely in assembly language (unlike MASM). **(2)** Editor, assembler and source file always in memory so you can go instantly from editing to assembling and back. **(3)** Eliminates the time needed to LINK programs. Executable .COM files can be created directly. (Also creates .OBJ files completely compatible with the IBM linker).

FAST ASSEM-86 **IS EASIER TO USE:**

FASM includes many other features to make your programming simpler.

- Listings are sent directly to screen or printer. Assemblies can be single stepped and examined without having to leave the editor.
- Access the built in cross reference utility from the editor.
- Full support of 186 and 286 (real mode) instructions.
- Both Microsoft and 8087 floating point formats are supported. 8087 and 287 instructions supported directly without macros for faster assembly.
- Calculator mode: Do math in any radix even using symbols from the symbol table.
- Direct to memory assembly feature lets you test execute your code from editor.
- Coming soon: A coordinated symbolic debugger.

COMPATIBILITY: ***FASM*** is designed for source code compatibility with MASM and supports most of its important features.

Introductory Price $199

Speedware™

Dealer inquires welcome
916-966-6247
Box D1, 2931 Northrop Avenue
Sacramento, CA 95825

IBM, Microsoft trademarks of IBM Corp., Microsoft Corp. respectively.

Circle **no. 97** on reader service card.

Listing Seven

```
PROGRAM Benchmark1 (INPUT, OUTPUT);
{ Sieve of Eratosthenes }

USES StopWatch;

CONST
  iterations = 10; { number of times benchmark executed }

VAR
  i: INTEGER;
{************************************************************}
PROCEDURE Sieve;

CONST
  size = 8190;

VAR
  i, prime, k, count: INTEGER;
  flags: ARRAY [0..size] OF BOOLEAN;

BEGIN
  count := 0;
  FOR i := 0 TO size DO BEGIN
    flags[i] := TRUE;
  END;
  FOR i := 0 TO size DO BEGIN
    IF flags[i] THEN BEGIN
      prime := i + i + 3;
      k := i + prime;
      WHILE k <= size DO BEGIN
        flags[k] := FALSE;
        k := k + prime;
      END;
      count := count + 1;
    END;
  END;
END; {Sieve}

{************************************************************}

BEGIN
  Start;
  FOR i := 1 TO iterations DO BEGIN
    Sieve;
  END;
  Stop;
END. {Benchmark1}
```

End Listing Seven

Listing Eight

```
MODULE Benchmark2;
(* Fibonacci series *)

FROM StopWatch IMPORT
  Start,
  Stop;
```

(Listing Eight begins on page 90)

Circle **no. 39** on reader service card.

Circle **no. 59** on reader service card.

Circle **no. 68** on reader service card.

Listing Eight

```
CONST
  iterations = 10; (* number of times benchmark executed*)
  length = 24; (* length of series *)

VAR
  i: INTEGER;
  useless: CARDINAL;

(*********************************************)

PROCEDURE Fibonacci (n: CARDINAL): CARDINAL;

BEGIN
  IF n > 2 THEN
    RETURN Fibonacci(n-1) + Fibonacci(n-2);
  ELSE
    RETURN 1;
  END;
END Fibonacci;

(*********************************************)

BEGIN
  Start;
  FOR i := 1 TO iterations DO
    useless := Fibonacci (length);
  END;
  Stop;
END Benchmark2.
```

End Listing Eight

Listing Nine

```
PROGRAM Benchmark2 (INPUT, OUTPUT);
{ Fibonacci Series }

USES StopWatch;

CONST
  iterations = 10; { number of times benchmark executed }
  length = 24; { length of series computed }

VAR
  i: INTEGER;

{************************************************************}

FUNCTION Fibonacci (n: WORD): WORD;

BEGIN
  IF n > 2 THEN BEGIN
    Fibonacci := Fibonacci(n-1) + Fibonacci(n-2);
  END ELSE BEGIN
    Fibonacci := 1;
  END;
END; {Fibonacci}
```

(Continued on page 92)

 (Listing Continued, text begins on page 74)

Listing Nine

```
{*************************************************************}

BEGIN
  Start;
  FOR i := 1 TO iterations DO BEGIN
    EVAL (Fibonacci (length));
  END;
  Stop;
END. {Benchmark2}
```

End Listing Nine

Listing Ten

```
MODULE Benchmark3;
(* text filter *)

FROM StopWatch IMPORT
  Start,
  Stop;

FROM FileSystem IMPORT
  Response,
  File,
  Lookup,
  Reset,
  Close,
  ReadByte,
  WriteByte;

FROM InOut IMPORT
  WriteString,
  WriteLn;

CONST
  iterations = 10;
  FileSize = 1000;

VAR
  i: INTEGER;

(******************************************)

PROCEDURE Init;

VAR
  i: INTEGER;
  f1: File;

BEGIN
  Lookup (f1, 'DK:file1.doc', TRUE);
  FOR i := 1 TO FileSize DO
    WriteByte (f1, 'a');
  END;
  Close (f1);
END Init;
```

```
(*****************************************)

PROCEDURE Filter;

VAR
  f1, f2: File;
  c: CHAR;

BEGIN
  Lookup (f1, 'DK:file1.doc', FALSE);
  IF NOT (f1.res = done) THEN
    WriteString ('file1.doc not found');
        WriteLn;
 END;
 Lookup (f2, 'DK:file2.doc', TRUE);
 LOOP
     ReadByte (f1, c);
     IF f1.eof THEN
       EXIT;
     END;
     (* process c *)
     WriteByte (f2, c);
   END;
   Close (f1);
   Close (f2);
END Filter;

(*****************************************)

BEGIN
  Init;
  Start;
  FOR i := 1 TO iterations DO
    Filter;
  END;
  Stop;
END Benchmark3.
```

End Listing Ten

Listing Eleven

```
PROGRAM Benchmark3 (INPUT, OUTPUT);
{ text filter }

USES StopWatch;

CONST
  iterations = 10;
  FileSize = 1000;

VAR
  i: INTEGER;

{***********************************}

PROCEDURE Init;

VAR
  i: INTEGER;
  f1: file of BYTE;

BEGIN
  ASSIGN (f1, 'file1.doc');
  REWRITE (f1);
  FOR i := 1 TO FileSize DO BEGIN
    WRITE (f1, ORD('a'));
  END;
  CLOSE (f1);
END; {Init}

{***********************************}

PROCEDURE Filter;

VAR
  f1, f2: file of BYTE;
  c: BYTE;

BEGIN
  ASSIGN (f1, 'file1.doc');
  ASSIGN (f2, 'file2.doc');
  RESET (f1);
  REWRITE (f2);
  WHILE NOT EOF (f1) DO BEGIN
    READ (f1, c);
    { process c }
    WRITE (f2, c);
  END;
  CLOSE (f1);
  CLOSE (f2);
END; {Filter}

{***********************************}

BEGIN
  Init;
  Start;
  FOR i := 1 TO iterations DO BEGIN
    Filter;
  END;
  Stop;
END. {Benchmark3}
```

End Listings

THE ANATOMY OF A

INTRODUCING
Interface Technologies' Modula-2 Software Development System

The computer press is hailing Modula-2 as "the next standard in programming languages." Modula-2 combines the strengths of Pascal with the features that made C so popular, like independent compilation and direct hardware control.

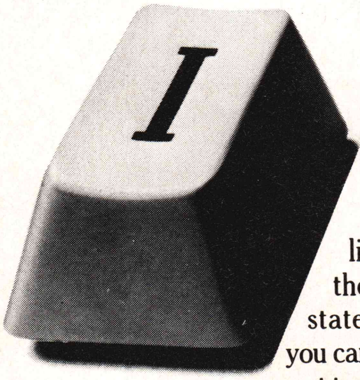
But until today, no company offered a Modula-2 system that made the development of software fast, easy and efficient. Now, though, there's a new tool at your disposal.

The fast, powerful tool for programmers

The breakthrough is here: Interface Technologies' new Modula-2 Software Development System for the IBM® PC, XT, AT and compatible computers to give programmers the same quantum leap in productivity spreadsheets and word processors gave to end-users. It can reduce monotonous wait time, will dramatically increase speed, help stop thoughtless mistakes, and free you to become more creative in virtually all of your programming efforts.

How to speed input and eliminate 30% of errors

Thirty percent of programming mistakes are syntax errors and simple typos in the program structure. Our "syntax-directed" Modula-2 editor does away with these time-consuming headaches once and for all.



Enter complete statements with one keystroke.

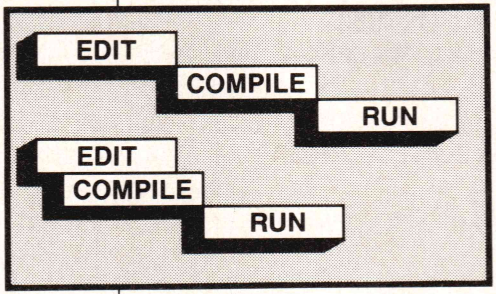
It speeds input by cutting manual typing as much as 90%, letting you enter statements with a single keystroke. For example, if you type a capital "I" to begin a line, the editor completes the logical "IF THEN" statement automatically, so you can concentrate on what you want to program, rather than concentrate on what you're typing.

The editor locks out errors, finishing statements and procedures in perfect accord with the standardized rules of Modula-2. It also indents and formats your text automatically, making programs easy to read and maintain, an important feature on big projects.

And if you leave an undefined variable or data type, the editor detects the mistake and gives you the option of on-line "help" to correct it. No other programming text editor offers you so much innovation at any price.

How to turn "wait time" into "work time"

The vast majority of programming time is spent waiting, and the biggest slowdown is most often with compilers.



It not only has a faster compiler, it also saves time by compiling while you edit.

Our compiler turns wait time to work time with a new innovation that lets you compile in the "background."

With background compilation, your program is automatically compiled into object code line by line as you work, every minute you spend writing or editing a Modula-2 program!

When you're finished editing, all that's left for the compiler is a quick mopping up job that generates optimized native code in a single pass.

How quick is "quick"?

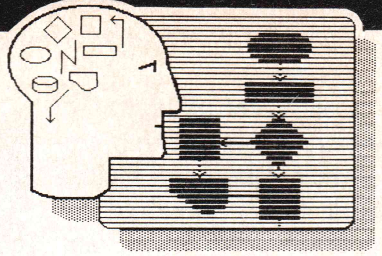
Thanks to background compilation and the fact that the compiler itself is so fast, Interface Technologies' compiler turns 100 lines of typical Modula-2 text into optimized machine code in *under five seconds.*

Plus the Interface compiler produces compact code with execution speed superior to that produced by any other Modula-2 compiler on the market.

How to do two things at once

Along with the background compiler and syntax-directed editor, which can save you hours every day and make you more productive, Interface Technologies' Software Development System gives your monitor

THE SOFTWARE DESIGNER



by Michael Swaine

When this magazine was first published in January 1976, its title was *Dr. Dobb's Journal of Tiny BASIC Calisthenics and Orthodontia*, with the subtitle "Running Light Without Overbyte" (a phrase that one reader recently asked us to bring back). The first issue was filled with very tight code (and discussions of code) designed to fit in the limited memory of the first microcomputers. Hence "running light"; hence "without overbyte": it made sense in 1976 to squeeze every last byte out of your system. But now it's nine years and ninety-nine issues of *DDJ* later, and the memory space of the typical microcomputer has increased a thousandfold. Does it make any sense to talk about running light in 1985?

To get an answer to that question, I called Dennis Allison, the paradigmatic light runner, in Geneva.

Dennis is the "Do" of "Dobb." When and he and Bob Albrecht of *People's Computer Company* launched this magazine as a three-issue newsletter, they gave the job of naming their creation to Rick Bakalinsky, who was then *PCC*'s entire production department. Rick somehow got the idea that Dennis was "Don" and combined "Don" and "Bob" to produce "Dobb" (doubling the "b" to make it more namelike).

But Dennis is not just Do; among other things (*many* other things), he is a visiting lecturer in Geneva this month and next month will be doing something or other in Crete. Among the memorable things Dennis has done for *DDJ* over the years is a column on algorithms he wrote during the editorial tenure of Suzanne Rodriguez (whose reminiscence about Bob and Dennis appears in the "Festschrift for Dr. Dobb" elsewhere in this issue).

Dennis readily agreed that the world of microcomputer programming had changed radically since the early days.

"It's different now that memory is free," he told me. "When *DDJ* started, the Altair had 256 bytes of memory. That's a far cry from today, when for a reasonable sum of money you can get 64K of memory, and many machines have half a megabyte. That changes the whole scale of things."

Did that mean that the urge to create the great hack was no longer there? In *Hackers*, Steven Levy wrote about "the last hacker"; was he right in his assessment that commercial considerations had nudged out this motivation?

"It's still there," Dennis said. "There's a certain pride in making code as tight and as efficient as possible. I write compilers and I take pleasure in coding as efficiently as I can in writing [parts of the] compiler. But often it is done as an aesthetic exercise more than anything else. If you look at Don Knuth's code, you'll find that it's elegant and well-organized—and efficient in this same sense."

But Knuth is an academician and an artist in his work. He recently got into typefont design for purely aesthetic reasons. Is tight coding professionally important, or is it just pretty?

"It's still important to use code efficiently. The one thing that's virtually guaranteed is that you won't have enough cycles. You'll never have enough memory, because expectations [can] rise faster than costs [can] drop."

Was it, then, a question of where you invested your hacking energy?

"Yes. The rule is that 4% of the code does 50% of the work, and the place where you need to squeeze code and get the number of instructions down is in that 4%."

Tightening the critical algorithms till they squeak, rather than scrounging space for one more cell in the spreadsheet?

Dennis agreed, and then went on to talk about something near and dear to the hearts of *DDJ* readers for nine years: the virtue of publishing program listings.

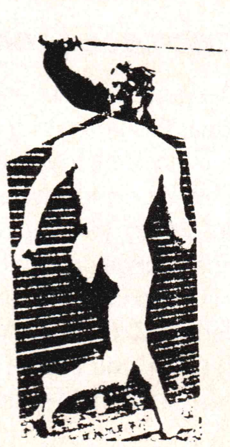
"*DDJ* is important as a journal because of the code. It was started to publish code and it has always published code. It publishes more code than any other magazine. And it's still my belief that people [programmers] learn from reading other people's code."

What they learn, of course, is how to program well: they pick up tricks, insights, algorithms, all of which are particularly well expressed when presented in the form in which they will ultimately be realized: as code. We hope to go right on publishing useful and educational code, including both programs significant in themselves and bits of code that demonstrate some exemplary algorithm or insight.

Toward the latter end, we will be reintroducing next month a column on algorithms, written by none other than Dennis Allison. The tentative title is "Allison on Algorithms." Well, it's descriptive.

With Bob Albrecht's column on "Realizable Fantasies," previewed in the "Festschrift for Dr. Dobb" elsewhere in this issue, the founders have returned to *DDJ*. **DDJ**

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 196**.



February 1985, #100

Dr. Dobb's Journal

Reader Service Card

A. Your primary job function: (Check one only)

- ☐ Company management (Pres., V.P., Treas., Owner, Gen. Mgr., Mktg. Dir.)
- ☐ Computer systems management (V.P. EDP., MIS Director, Data Processing Mgr., Data Communications Mgr., Network Planner)
- ☐ Engineering management (V.P. Engrg., Chief Engr., Tech. Director, Dir. R&D)
- ☐ Systems integrators (Systems Designer, Project Engr., Systems Application Engr., Technical Staff Members)
- ☐ Consultants (Computer/EDP/Data Communications Consultant)
- ☐ Educators (Educational users and instructors of computer technology)
- ☐ Systems/Programming specialists—mini-micro systems
- ☐ Other (Please specify)__________________

B. Which languages are you MOST interested in?

☐ BASIC	☐ C	☐ PL/1
☐ Fortran	☐ LISP	☐ APL
☐ COBOL	☐ Prolog	☐ Logo
☐ Pascal	☐ Ada	☐ Smalltalk
☐ Modula-2	☐ Forth	Other ________

C. What is the operating system?

- ☐ CP/M (or derived)
- ☐ UNIX (or derived)
- ☐ MS-DOS (or derived)
- ☐ Other

D. Please indicate which of the following microcomputers you currently own and/or plan to buy in the next 12 months.

	Own	Plan to Buy
Apple	☐	☐
Commodore	☐	☐
Digital Equipment/DEC	☐	☐
Heath/Zenith	☐	☐
Hewlett-Packard	☐	☐
IBM	☐	☐
Macintosh	☐	☐
Radio Shack/Tandy TRS 80	☐	☐
Texas Instruments	☐	☐
Other (Specify)	☐	☐
None	☐	☐

E. What best describes the work you do with this microcomputer?

- ☐ Business functions
- ☐ Software/Hardware development
- ☐ Scientific/Engineering/R&D applications

F. Are you the decision maker or very influential in computer-related purchases at work?

☐ Yes ☐ No

G. Is your company a dealer, distributor, or systems house for microcomputers?

☐ Yes ☐ No

H. Subscriber

☐ Yes ☐ No

To obtain information about products or services mentioned in this issue, circle the appropriate number listed below. Use bottom row to vote for best article in issue. One card per person.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 24 25 26 27
28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81
82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108
109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135

Articles: 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214

This offer good until May 31, 1985

Name ______________________________ **Phone (____)**____________

Address ____________________________________

City/State/Zip ________________________________

February 1985, #100

Dr. Dobb's Journal

Reader Service Card

A. Your primary job function: (Check one only)

- ☐ Company management (Pres., V.P., Treas., Owner, Gen. Mgr., Mktg. Dir.)
- ☐ Computer systems management (V.P. EDP., MIS Director, Data Processing Mgr., Data Communications Mgr., Network Planner)
- ☐ Engineering management (V.P. Engrg., Chief Engr., Tech. Director, Dir. R&D)
- ☐ Systems integrators (Systems Designer, Project Engr., Systems Application Engr., Technical Staff Members)
- ☐ Consultants (Computer/EDP/Data Communications Consultant)
- ☐ Educators (Educational users and instructors of computer technology)
- ☐ Systems/Programming specialists—mini-micro systems
- ☐ Other (Please specify)__________________

B. Which languages are you MOST interested in?

☐ BASIC	☐ C	☐ PL/1
☐ Fortran	☐ LISP	☐ APL
☐ COBOL	☐ Prolog	☐ Logo
☐ Pascal	☐ Ada	☐ Smalltalk
☐ Modula-2	☐ Forth	Other ________

C. What is the operating system?

- ☐ CP/M (or derived)
- ☐ UNIX (or derived)
- ☐ MS-DOS (or derived)
- ☐ Other

D. Please indicate which of the following microcomputers you currently own and/or plan to buy in the next 12 months.

	Own	Plan to Buy
Apple	☐	☐
Commodore	☐	☐
Digital Equipment/DEC	☐	☐
Heath/Zenith	☐	☐
Hewlett-Packard	☐	☐
IBM	☐	☐
Macintosh	☐	☐
Radio Shack/Tandy TRS 80	☐	☐
Texas Instruments	☐	☐
Other (Specify)	☐	☐
None	☐	☐

E. What best describes the work you do with this microcomputer?

- ☐ Business functions
- ☐ Software/Hardware development
- ☐ Scientific/Engineering/R&D applications

F. Are you the decision maker or very influential in computer-related purchases at work?

☐ Yes ☐ No

G. Is your company a dealer, distributor, or systems house for microcomputers?

☐ Yes ☐ No

H. Subscriber

☐ Yes ☐ No

To obtain information about products or services mentioned in this issue, circle the appropriate number listed below. Use bottom row to vote for best article in issue. One card per person.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 24 25 26 27
28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81
82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108
109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135

Articles: 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214

This offer good until May 31, 1985

Name ______________________________ **Phone (____)**____________

Address ____________________________________

City/State/Zip ________________________________

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT #27346, PHILADELPHIA, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

SOFTWARE TOOLS FOR ADVANCED PROGRAMMERS
Dr. Dobb's Journal

P.O. BOX 13851
PHILADELPHIA, PA 19101

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT #27346, PHILADELPHIA, PA.

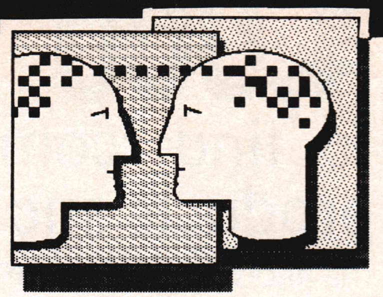
POSTAGE WILL BE PAID BY ADDRESSEE

SOFTWARE TOOLS FOR ADVANCED PROGRAMMERS
Dr. Dobb's Journal

P.O. BOX 13851
PHILADELPHIA, PA 19101

Circle **no. 93** on reader service card.

Circle **no. 15** on reader service card.



by Robert Blum

Buffered Disk I/O

The attempt to predict the length of a mechanical operation is easy when the system involves only a single device. The level of complication increases dramatically, however, when the system comprises two or more dependent devices. It simplifies the matter to realize that total system performance will be no greater than that of the slowest part.

A personal computer is a perfect example. The electronic portion of the computer performs its task at blinding speeds, but read a record from disk, and you quickly find the slowest part of the computer. A few techniques can minimize the mechanical delays encountered when transferring data to a disk drive. The most common examples are logical sector buffering and physical sector skewing. Both are typically a standard part of the operating system.

Further improvements are more difficult to achieve. Most require additional or improved hardware, which is impractical in many cases. This places the burden of any further improvement directly on the application program.

To increase the throughput of the CP/M, we can use any extra memory to buffer as much data as is practical. This manner of speeding things up circumvents one of CP/M's problem areas and puts all the system's resources to use—probably for the first time.

The CP/M problem has to do with the number of I/O operations necessary to write a record to disk. This was of no concern when disks were recorded in single-density format. But as improved hardware enabled higher bit rates, reliably larger sector sizes saw almost immediate use. This change allowed the total storage capacity of the media to double. There was, however, a price to be paid for this improvement.

CP/M uses a single I/O buffer to transfer data between memory and the disk drive. A 1:1 logical-to-physical

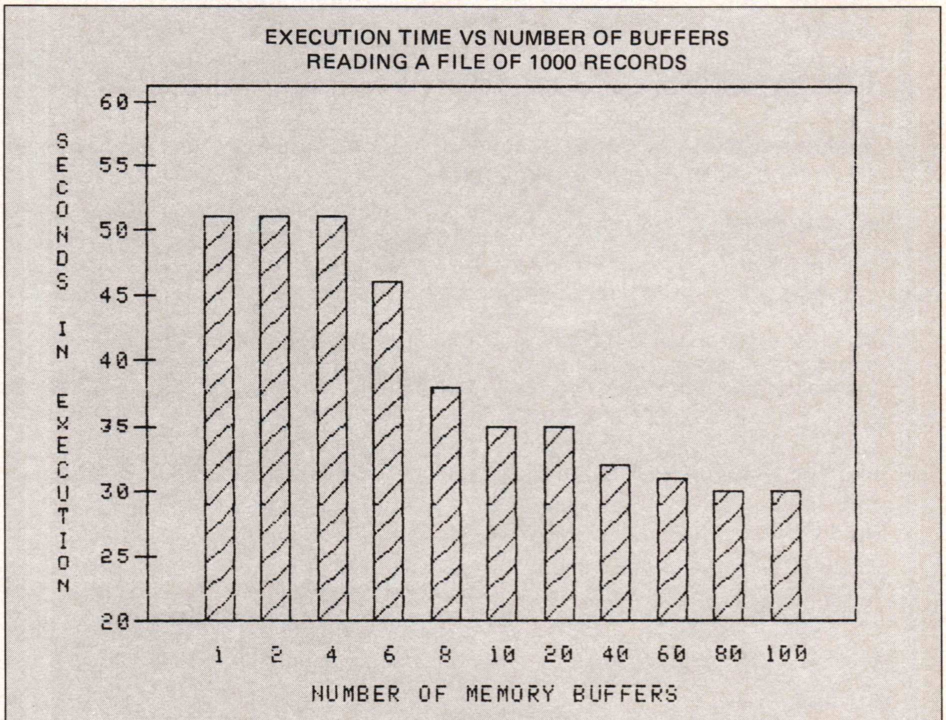


Figure 1

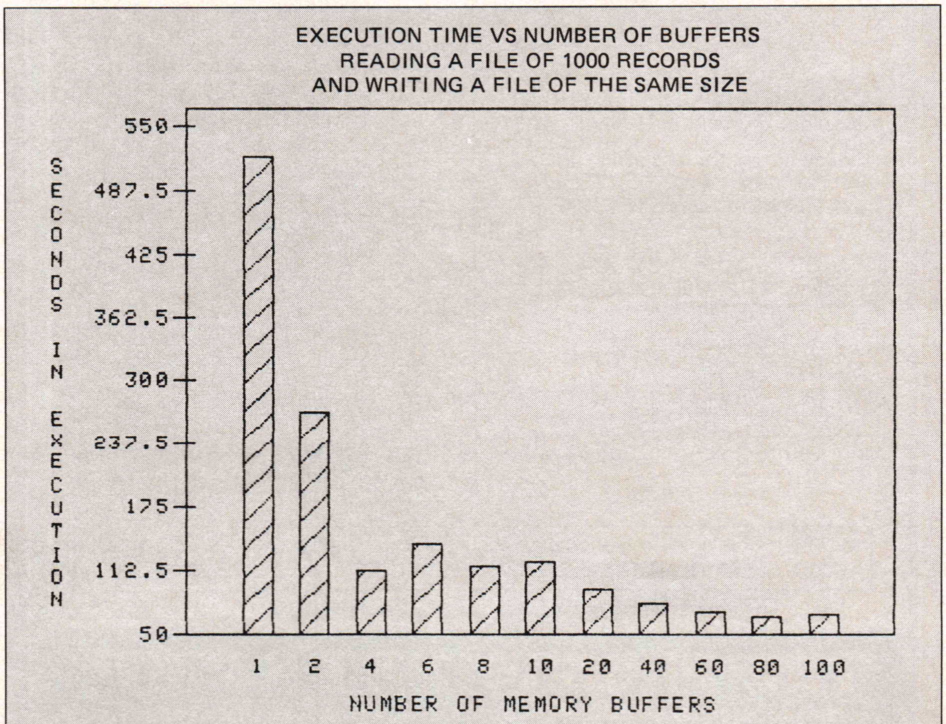


Figure 2

sector size relationship ensures transfer without buffer conflict. Changing the relationship to 4:1, however, can require up to three times as much disk I/O to transfer one logical sector of data. For a more complete explanation of buffer conflicts, refer to the December column.

Putting memory-resident data buffers to work speeding data through your machine can be as simple as specifying a parameter. When writing CBASIC as a Master's project, Gordon Eubanks made provision for the programmer to indicate how many data buffers to allocate. The last parameter of the OPEN statement (following the BUFF keyword) specifies how many record areas to set aside.

To plot the effect of using a varying number of data buffers against program execution time, I wrote two simple benchmark programs. The first (Listing One, below) requests how many data buffers to allocate, opens an input file, and executes 1000 sequential reads. Total program execution time is plotted in Figure 1 (page 98). The system I used to run the benchmark utilizes a physical buffer size of 512 bytes, which represents a logical-to-physical sector ratio of 4:1.

As you can see from Figure 1, there was no runtime improvement when from one to four buffers were allocated. When six buffers were used, however, runtime decreased markedly. An even larger decrease resulted when eight or more buffers were allocated.

The reason for the runtime improvement is that the physical sector buffer contains four logical sectors. In operation, the first program read request fills the physical sector buffer with four logical sectors. Each subsequent read request is satisfied directly from the memory buffer until it must be replenished from disk. When the data buffer exceeds the size of one physical sector, the possibility of reading two sectors in a single revolution of the disk is very good. Allocating six or more buffers reduced the program runtime by nearly 50%—obviously an improvement well worth pursuing.

Runtime improvements come even more quickly when a program progressively reads from one file and writes to another. To plot this condition, I wrote a second benchmark program (Listing 2, page 100). Its results are contained in Figure 2 (page 98). The effect of multiple data buffers on this scenario is even more dramatic.

The reason for the sudden decrease in runtime is the single I/O buffer maintained by CP/M. When only a single record buffer is chosen for the benchmark, most of the program's runtime is consumed by buffer maintenance. To understand better why this function consumes so much time, let's trace CP/M's data flow.

The first record read requires that the system perform a physical sector read from disk. The next I/O operation is a write. Because CP/M's buffer contains only data from a previous read, you can overwrite it without regard to destroying data. The next operation is another read. But before the selected physical sector can be brought in from the disk, the current CP/M buffer must be written onto disk; otherwise, the data it contains will be overwritten. The next operation is another write, but first the physical sector written onto disk during the last write operation must be read back into memory for updating.

From this study of a typical data flow, we realize that each write operation includes a pre-read of the physical sector on all but the first logical sector of each physical sector. Also included in each read is a physical sector read to refresh CP/M's I/O buffer. Data buffering of only a single record in this situation can account for an I/O reduction of at least 50%.

To conclude this discussion, I want to share an assembly language routine (Listing Three, page 102) that also performs the data buffering function. SFXIO is a part of SYSLIB3 now available in the public domain and from Echelon, Inc. One particularly interesting concept used in this routine is the transfer of information between the application program and the subroutine by use of a control block. This method is especially effective in trimming the development cycle because it clusters all the relevant control information into a fixed area of memory.

I will leave any further discussion of this subroutine to you. If you have any questions about this subroutine, please write me in care of this column. If you want to see about getting SYSLIB3 on disk, you can reach Echelon at the following address: 101 First Street, Los Altos, CA 94022 (415) 948-3820.

DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 197**.

CP/M Exchange

Listing One

```
REM
REM THIS PROGRAM READS A FILE OF 1000 RECORDS WITH VARYING NUMBERS
REM OF MEMORY BUFFERS FOR THE PURPOSE OF SPEED COMPARISIONS
REM

100.0
        STRING  DRIVE,\
                FILE.NAME,\
                INPUT.DATA

        INTEGER RECORD.SIZE,\
                FILE.NUMBER,\
                RECORD.COUNT,\
                LOOP.COUNTER,\
                NO.OF.BUFFERS
```

(Continued on next page)

Listing One

```
        DRIVE = "C:"
        FILE.NAME = "SPEED.DAT"
        RECORD.SIZE = 128
        FILE.NUMBER = 1
        RECORD.COUNT = 1000
        NO.OF.BUFFERS = 1

200.0
        INPUT "Enter number of input buffers to use: "; NO.OF.BUFFERS
        OPEN DRIVE + FILE.NAME\
                RECL RECORD.SIZE\
                AS FILE.NUMBER\
                BUFF NO.OF.BUFFERS

300.0
        FOR LOOP.COUNTER = 1 TO RECORD.COUNT
        READ #FILE.NUMBER; INPUT.DATA
        NEXT LOOP.COUNTER
        CLOSE FILE.NUMBER
        END

                        1   - 51
                        2   - 51  R S
                        4   - 51  U E
                    M B 6   - 46  N C
                    E U 8   - 38  T O
                    M F 10  - 35  I N
                    O F 20  - 35  M D
                    R E 40  - 32  E S
                    Y R 60  - 31
                      S 80  - 30
                        100 - 30

              LISTING 1 - READ ONLY SPEED TEST
```

End Listing One

Listing Two

```
REM
REM THIS PROGRAM READS AND WRITES A FILE OF 1000 RECORDS WITH VARYING
REM NUMBERS OF MEMORY BUFFERS FOR THE PURPOSE OF SPEED COMPARISIONS
REM

100.0

        STRING  DRIVE,\
                INPUT.FILE.NAME,\
                OUTPUT.FILE.NAME,\
                INPUT.DATA

        INTEGER RECORD.SIZE,\
                INPUT.FILE.NUMBER,\
                OUTPUT.FILE.NUMBER,\
                RECORD.COUNT,\
                LOOP.COUNTER,\
                NO.OF.BUFFERS

        DRIVE = "C:"
        INPUT.FILE.NAME = "SPEED.DAT"
        OUTPUT.FILE.NAME = "OUTSPEED.DAT"
        RECORD.SIZE = 128
        INPUT.FILE.NUMBER = 1
        OUTPUT.FILE.NUMBER = 2
        RECORD.COUNT = 1000
        NO.OF.BUFFERS = 1

200.0
        INPUT "Enter number of input buffers to use: "; NO.OF.BUFFERS
        OPEN DRIVE + INPUT.FILE.NAME\
                RECL RECORD.SIZE\
                AS INPUT.FILE.NUMBER\
                BUFF NO.OF.BUFFERS
        CREATE DRIVE + OUTPUT.FILE.NAME\
                RECL RECORD.SIZE\
```

(Continued on page 102)

Listing Two

```
              AS OUTPUT.FILE.NUMBER\
              BUFF NO.OF.BUFFERS

300.0
        FOR LOOP.COUNTER = 1 TO RECORD.COUNT
        READ #INPUT.FILE.NUMBER; INPUT.DATA
        PRINT #OUTPUT.FILE.NUMBER; INPUT.DATA
        NEXT LOOP.COUNTER
        CLOSE INPUT.FILE.NUMBER, OUTPUT.FILE.NUMBER
        END
                                          1    520
                                          2    269
                                   M B    4    113  R S
                                   E U    6    138  U E
                                   M F    8    117  N C
                                   O F    10   123  T O
                                   R E    20   94   I N
                                   Y R    40   80   M D
                                     S    60   73   E S
                                          80   68
                                          100  69

                        LISTING 2 - READ AND WRITE SPEED TEST
```

End Listing Two

Listing Three

```
;
; SYSLIB Module Name:  SFXIO
; Author:  Richard Conn
; SYSLIB Version Number:  3.0
; Module Version Number:  1.0
; Module Entry Points:
;       FX$GET          FX$PUT
;
;       SFXIO provides a group of routines which can perform byte-oriented
; file I/O with a user-defined buffer size.  All of these routines work with
; an I/O Control Block which is structured as follows:
;
;               Byte    Length (Bytes)  Function
;                 0             1       Number of 128-byte pages in
;                                               working buffer (set by user)
;                 1             1       End of File Flag (set and used
;                                               by SFXIO)
;                 2             2       Byte Counter (set and used by SFXIO)
;                 4             2       Next Byte Pointer (set and used by
;                                               SFXIO)
;                 6             2       Address of Working Buffer (set by user)
;                 8            36       FCB of File (FN and FT Fields set by
;                                               user, rest set by SFXIO)
;
;       The following DB/DS structure can be used in the calling program
; to implement the I/O Control Block:
;
;       IOCTL:
;               DB      8       ; Use 8 128-byte pages (1K)
;               DS      1       ; Filled by SFXIO
;               DS      2       ; Filled by SFXIO
;               DS      2       ; Filled by SFXIO
;               DW      WORKBF  ; Address of Working Buffer
;       IOCFCB:
;               DB      0       ; Current Disk (Inited by SFXIO)
;               DB      'MYFILE  '      ; File Name
;               DB      'TXT'           ; File Type
;               DS      24      ; Fill Out 36 Bytes
;       WORKBF: DS      1024    ; Working Buffer
;
;       All uses of the routines contain the address of IOCTL in DE.
; Note that if you use a buffer for input, DO NOT use it for output also!
;
```

(Continued on page 106)

CP/M Exchange (Listing Continued, text begins on page 98)

Listing Three

```
;
;  EXTERNAL SYSLIB REFERENCES
;
        EXT     F$DELETE,F$MAKE,F$OPEN,F$CLOSE
        EXT     FXI$OPEN,FXO$OPEN,FXI$CLOSE,FXO$CLOSE
        EXT     F$READ,F$WRITE
        EXT     INITFCB
        EXT     HMOVB,SHFTRH,SHFTLH

;
;  CONSTANTS
;
TBUFF   EQU     80H       ; DMA BUFFER
CTRLZ   EQU     'Z'-'@'  ; ^Z

;
;  MACRO ROUTINES FOR FXIO
;
PUTRG   MACRO
        PUSH    B
        PUSH    D
        PUSH    H
        ENDM

GETRG   MACRO
        POP     H
        POP     D
        POP     B
        ENDM

;
;  ENTRY POINT TO READ NEXT BUFFER FULL FROM DISK
;
FXIO0:
        LDA     EOF       ; CHECK FOR EOF
        ORA     A         ; ABORT IF ALREADY AT EOF
        RNZ               ; NZ IS ERROR CODE
        LHLD    FCB       ; PT TO FCB
        XCHG              ; ... IN DE
        LDA     BCNT      ; GET BLOCK COUNT
        MOV     B,A
        LHLD    BUFADR    ; GET ADDRESS OF BUFFER
FXIO1:
        PUSH    H         ; SAVE BUFFER PTR
        LHLD    FCB       ; PT TO FCB
        XCHG              ; ... IN DE
        CALL    F$READ    ; READ NEXT BLOCK (SECTOR)
        POP     H         ; GET PTR TO BUFFER
        JNZ     FXIO2
        LXI     D,TBUFF   ; COPY INTO MEMORY BUFFER
        PUSH    B         ; SAVE COUNTER
        XCHG
        MVI     B,128
        CALL    HMOVB
        XCHG
        POP     B
        DCR     B         ; COUNT DOWN
        JNZ     FXIO1
        MVI     A,0       ; SET NO EOF
        JMP     FXIO3
FXIO2:
        MVI     A,0FFH    ; SET EOF
FXIO3:
        STA     EOF       ; SET EOF FLAG
        LHLD    BUFADR    ; PT TO FIRST BYTE AS NEXT BYTE
        SHLD    BYTENXT
        LDA     BCNT      ; SET BLOCK COUNT

        SUB     B         ; ADJUST
        MOV     H,A       ; RESULT IN HL
        MVI     L,0
```

(Continued on page 109)

DDJ BACK ISSUES

#69 Volume VII, No. 7:
IBM-PC Issue: CP/M-86 vs. MSDOS (A Technical Comparison)—Hi-Res Graphics on the IBM-PC—PDP-1802, Part II—Review of Word Processors for IBM.

#70 Volume VII, No. 8:
Argum "C" Command Line Processor—SEND/RECEIVE File Transfer Utilities—Intel's 8087 Performance Evaluation.

#71 Volume VII, No. 9:
FORTH Issue: Floating-Point Package—H-19 Screen Editor—Relocating, Linking Loader—Z8000 Forth—Forth Programming Style—8086 ASCII-Binary Conversion Routines—CP/M Conditional SUBMIT.

#72 Volume VII, No. 10:
Portable Pidgin for Z80—68000 Cross Assembler, Part I—MODEM and RCP/Ms—Simplified 68000 Mnemonics—Nested Submits—8086/88 Trig Lookup.

#73 Volume VII, No. 11:
Wildcard UNIX Filenames—Tests for Pidgin—68000 Cross Assembler Listing, Part 2—Adding More BDOS Calls—The Perfect Hash—BASIC Memory Management—Benchmarks for CP/M-86 vs. MSDOS, and the 8087.

#76 Volume VIII, Issue 2:
PISTOL, A Forth-like Portably Implemented Stack Oriented Language—Program Linkage by Coroutines. Forth to BASIC—Linking CP/M Functions to Your High-Level Program—Concurrent CP/M-86—CP/M-80 Expansion Card for the Victor 9000—REVAS Disassembler.

#77 Volume VIII, Issue 3:
The Augusta P-Code Interpreter—A Small-C Operating System—6809 Threaded Code: Parametrization and Transfer of Control—A Common-Sense Guide to Faster, Small BASIC—A Fundamental Mistake in Compiler Design—Basic Disk I/O, Part I.

#78 Volume VIII, Issue 4:
RECLAIM Destroyed Directories—Binary Magic Numbers—8080 Fig-Forth Directory & File System—SAY" Forth Votrax Driver—TRS-80 8080 to Z80 Translator—Basic Disk I/O, Part II.

#79 Volume VIII, No. 5:
The Augusta Compiler—A Fast Circle Routine—Enhancing the C Screen Editor—Shifts and Rotations on the Z80—The SCB, TSX, and TXS Instructions of the 6502 and 6800—MS-DOS vs. CP/M-86—Controlling MBASIC—The Buffered Keyboard—IBM PC Character Set Linker—Flip Utility for the IBM PC.

#80 Volume VIII, Issue 6:
Fast Divisibility Algorithms—B-Tree ISAM Concepts—CP/M BDOS AND BIOS Calls for C—Serial Expansion in Forth—Fast Matrix Operations in Forth, Part I—Yes, You Can Trace Through BDOS—Julian Dates for Microcomputers—8088 Addressing Modes—8088 Line Generator—CP/M Plus.

#81 Volume VIII, Issue 7:
The Augusta Compiler, continued—RED: A Better Screen Editor, Part I—Anatomy of a Digital Vector and Curve Generator—Fast Matrix Operations in Forth, Part II—The AGGHHHI Program—MBOOT Revisited—CP/M Plus Feedback—MS-DOS Rebuttal—68000 Tools—Sizing Memory on the IBM PC.

#82 Volume VIII, Issue 8:
Serial-to Parallel: A Flexible Utility Box—McWORDER: A Tiny Text Editor—And Still More Fifth Generation Computers—Specialist Symbols and I/O Benchmarks for CP/M Plus—CP/M Plus Memory Management—Zero Length File Test—PAUSEIF, QUITIF, and now SKIPIF—ACTxx Cross Assemblers.

#83 Volume VIII, No. 9:
FORTH ISSUE: Forth and the Motorola 68000—Nondeterministic Control Words in Forth—A 68000 Forth Assembler—GO in Forth—Precompiled Forth Modules—Signed Integer Division—Some Forth Coding Standards—The Forth Sort.

#84 Volume VIII, No. 10:
Unix to CP/M Floppy Disk File Conversion—A Small-C Help Facility—Attaching a Winchester Hard Disk to the S-100 Bus—Using Epson Bit-Plot Graphics—8086/88 Function Macros—Auto Disk Format Selection—CP/M Plus Device Tables.

#85 Volume VIII, Issue 11:
A Kernel for the MC68000—A DML Parser—Towards a More Writable Forth Syntax—Simple Graphics for Printer—Floating-Point Benchmarks.

#86 Volume VIII, Issue 12:
Faster Circles for Apples—Cursor Control for Dumb Terminals—Dysan's Digital Diagnositic Diskette—Interfacing a Hard Disk Within a CP/M Environment—The New MS-DOS EXEC Function.

#87 Volume IX, Issue 1:
A Structured Preprocessor for MBASIC—A Simple Window Package—Forth to PC-DOS Interface—Sorted Diskette Directory Listing for the IBM PC—Emulate WordStar on TOPS-20—More on optimizing compilers—The PIP mystery device contest.

#88 Volume IX, Issue 2:
Telecommunications Issue: Micro to Mainframe Connection—Communications Protocols—Unix to Unix Network Utilities—VPC: A Virtual Personal Computer for Networks—PABX and the Personal Computer—BASIC Language Telecommunications Programming—U.S. Robotics S-100 Card Modem.

#89 Volume IX, Issue 3:
RSA: A Public Key Cryptography System, Part I—Introduction to PL/C: Programming Language for Compilers—Program Design Using Pseudocode—More on Binary Magic Numbers—How fast is CP/M Plus?—CP/M 2.2 BIOS Function: SELDSK—The results of the Floating-Point benchmark.

#90 Volume IX, Issue 4:
Optimizing Strings in C—Expert Systems and the Weather—RSA: A Public Key Cryptography System, Part II—Several items on CP/M Plus, CP/M v2.2 Compatibility—BDOS Function 10: Vastly Improved—More on MS-DOS EXEC Function—Low-Level Input-Output in C.

#91 Volume IX, Issue 5:
Introduction to Modula-2 for Pascal Programmers—Converting Fig-Forth to Forth-83—Sixth Generation Computers—A New Library for Small-C—Solutions to Quirks in dBASE II.

#92 Volume IX, Issue 6:
CP/M on the Commodore 64—dBASE II Programming Techniques—First Chinese Forth: A Double-Headed Approach—cc-A Driver for a Small-C Programming System—A New Library for Small-C (Part II)—Comments on Sixth Generation Computers—Review of Turbo Pascal.

#93 Volume IX, Issue 7:
RSX under CP/M Plus—p-A Small-C Preprocessor—A Simple Minimax Algorithm—Languages and Parentheses (A Suggestion for Forth-like Languages)—Comments on assembly language development packages, RSX to patch CP/M 2.2 with CP/M, iRMS-86 for the IBM PC, C Programming Tools.

#94 Volume IX, Issue 8:
SCISTAR: Greek and Math Symbols with WordStar—A File Comparator for CP/M Plus—Designing a File Encryption System—A Small-C Concordance Generator.

#95 Volume IX, Issue 9:
Forth Special Issue!—File Maintenance in Forth—Forth and the Fast Fourier Transform—Computing with Streams—A Forth Native-Code Cross Compiler for the MC68000—The FVG Standard Floating-Point Extension—CP/M Plus: Interbank Memory Moves Without DMA—ways to make C more powerful and flexible.

#96 Volume IX, Issue 10:
More dBASE II Programming Techniques—Simple Calculations with Complex Numbers—GREP. C: A Unix-like Generalized Regular Expression Parser—An optimization scheme for compilers, MSDOS 2.0 Filters, Sizing RAM under MSDOS, Two programming systems illustrating Runge-Kutta integration.

#97 Volume IX, Issue 11:
Adding Primitive I/O Functions to muLISP—Program Monitor Package: Using Interrupts to Instrument Applications—CP/M 2.2 Goes PUBlic—A Guide to Resources for the C Programmer—RESORT.

#98 Volume IX, Issue 12:
Varieties of Unix—Unix Device Drivers—A Unix Internals Bibliography—A file Browser Program—An Introduction to Parsing.

#99 Volume X, Issue 1:
Fatten Your Mac—QuickDraw Meets Image Writer—Archiving Files with CP/M 80 and CP/M 86—MBOOT and MODEM for the C-64's CP/M—Unstructured Forth Programming: An Introduction.

TO ORDER:
Send $3.50 per issue to: **Dr. Dobb's Journal,** 2464 Embarcadero Way, Palo Alto, CA 94303

Please send me the issue(s) circled: **69 70 71 72**
73 76 77 78 79 80 81 82 83 84 85 86
87 88 89 90 91 92 93 94 95 96 97 98 99

I enclose $__________ (U.S. check or money order).
Outside the U.S., add $.50 per issue.

I have read the postal instructions and understand that I will not receive my order unless I have sent the correct payment amount.

Please charge my: ☐ Visa ☐ M/C ☐ Amer. Exp.

Card No. ____________________ Exp. Date________

Signature __________________________

Name __________________________

Address __________________________

City __________________________

State _____________ Zip _____________

Availability on first come/first serve basis. Outside the U.S. add $.50 per issue ordered. Price includes issue, handling, and shipment by second class or foreign surface mail. Within the U.S., please allow 6–9 weeks to process your order second class. For faster service within the U.S., we'll ship UPS if you *add* $1.00 for 1–2 issues and $.50 for *each issue* thereafter. We need a street address, not a P. O. Box. Airmail rates: To Canada *add* $1.75 *per magazine*, all other foreign *add* $3.00 *per magazine.*

Circle **no. 81** on reader service card. Code 100

Listing Three

```
        CALL    SHFTRH   ; SET COUNT
        SHLD    BYTECNT  ; SET BYTE COUNT
        XRA     A        ; SET NO ERROR
        RET

;
;   ENTRY POINT TO FLUSH BUFFER TO DISK AND SET UP FOR NEXT WRITE
;
FXOO0:
        LDA     BCNT     ; GET NUMBER OF BLOCKS
        MOV     B,A      ; ... IN B
        LHLD    BYTECNT  ; NUMBER OF BYTES YET TO GO
        CALL    SHFTLH   ; SHIFT INTO H
        MOV     A,B      ; COMPUTE NUMBER TO WRITE
        SUB     H
        MOV     B,A      ; COUNT IN B
        LHLD    BUFADR   ; PT TO FIRST BYTE
FXOO1:
        MOV     A,B      ; CHECK FOR DONE
        ORA     A        ; 0=DONE
        JZ      FXOO2
        DCR     B        ; COUNT DOWN
        PUSH    B        ; SAVE COUNT
        LXI     D,TBUFF  ; PT TO WRITE BUFFER
        MVI     B,128    ; WRITE 128 BYTES
        CALL    HMOVB    ; COPY INTO BUFFER
        XCHG
        LHLD    FCB      ; PT TO FCB
        XCHG
        CALL    F$WRITE  ; WRITE TO DISK NEXT BLOCK (SECTOR)
        POP     B        ; GET COUNT
        JZ      FXOO1
        XRA     A        ; SET ERROR CODE
        RET
;
;   ENTRY POINT TO INIT BUFFERS FOR NEXT WRITE
;
FXOO2:
        LHLD    BUFADR   ; SET NEXT BYTE
        SHLD    BYTENXT
        XRA     A        ; SET NO EOF
        STA     EOF
        LDA     BCNT     ; SET BLOCK COUNT
        MOV     H,A      ; ... IN HL
        MVI     L,0
        CALL    SHFTRH   ; SHIFT RIGHT ONE BIT
        SHLD    BYTECNT  ; SET BYTE COUNT
        MVI     A,0FFH   ; NO ERROR
        ORA     A
        RET

;
;   GET NEXT BYTE FROM BUFFER/FILE
;       ON INPUT, DE PTS TO IOCB
;       ON OUTPUT, A=CHAR AND Z FLAG SET IF PAST EOF
;
FX$GET::
        PUTRG            ; SAVE REGS
        CALL    PUTADR   ; PUT ADDRESSES
        LHLD    BYTECNT  ; GET REMAINING COUNT
        MOV     A,H      ; NO MORE BYTES?
        ORA     L
        JNZ     FXGET1
        CALL    FXIO0    ; READ NEXT BUFFER FULL
        JNZ     ERRET
FXGET1:
        LHLD    BYTENXT  ; PT TO NEXT BYTE
        MOV     A,M      ; GET IT
        STA     BYTE     ; SAVE IT
        INX     H        ; PT TO NEXT
        SHLD    BYTENXT  ; SET PTR
        LHLD    BYTECNT  ; COUNT DOWN
        DCX     H
```

(Continued on page 112)

Announcing BOUND VOLUME 7

Every 1982 Issue Available For Your Personal Reference.

Vol. 7 1982

In 1982 we introduced several significant pieces of software, including the RED text editor and the Runic extensible compiler, and we continued to publish utility programs and useful algorithms. Two new columns, The CP/M Exchange and The 16-Bit Software Toolbox, were launched, and we devoted special issues to FORTH and telecommunications. Resident Intern Dave Cortesi supplied a year of "Clinic" columns while delivering his famous review of JRT Pascal and writing the first serious technical comparison of CP/M-86 and MSDOS. This was also the year we began looking forward to today's generation of microprocessors and operating systems, publishing software for the Motorola 68000 and the Zilog Z8000 as well as Unix code. And in December, we looked beyond, in the provocative essay, "Fifth-generation Computers."

Vol. 1 1976

The material brought together in this volume chronicles the development in 1976 of Tiny BASIC as an alternative to the "finger blistering," front-panel, machine-language programming which was then the only way to do things. This is always pertinent for bit crunching and byte saving, language design theory, home-brew computer construction and the technical history of personal computing.
Topics include: Tiny BASIC, the (very) first word on CP/M, Speech Synthesis, Floating Point Routines, Timer Routines, Building an IMSAI, and more.

Vol. 2 1977

1977 found **DDJ** still on the forefront. These issues offer refinements of Tiny BASIC, plus then state-of-the-art utilities, the advent of PILOT for microcomputers and a great deal of material centering around the Intel 8080, including a complete operating system. Products just becoming available for reviews were the H-8, KIM-1, MITS BASIC, Poly Basic, and NIBL. Articles are about Lawrence Livermore Lab's BASIC, Alpha-Micro, String Handling, Cyphers, High Speed Interaction, I/O, Tiny Pilot & Turtle Graphics, many utilities, and even more.

Vol. 3 1978

The microcomputer industry entered its adolescence in 1978. This volume brings together the issues which began dealing with the 6502, with mass-market machines and languages to match. The authors began speaking more in terms of technique, rather than of specific implementations; because of this, they were able to continue laying the groundwork industry would follow. These articles relate very closely to what is generally available today. Languages covered in depth were SAM76, Pilot, Pascal, and Lisp, in addition to RAM Testers, S-100 Bus Standard Proposal, Disassemblers, Editors, and much, much more.

Vol. 4 1979

This volume heralds a wider interest in telecommunications, in algorithms, and in faster, more powerful utilities and languages. Innovation is still present in every page, and more attention is paid to the best ways to use the processors which have proven longevity—primarily the 8080/Z80, 6502, and 6800. The subject matter is invaluable both as a learning tool and as a frequent source of reference.
Main subjects include: Programming Problems/Solutions, Pascal, Information Network Proposal, Floating Point Arithmetic, 8-bit to 16-bit Conversion, Pseudo-random Sequences, and Interfacing a Micro to a Mainframe—more than ever!

Vol. 5 1980

All the ground-breaking issues from 1980 in one volume! Systems software reached a new level with the advent of CP/M, chronicled herein by Gary Kildall and others (**DDJ's** all-CP/M issue sold out within weeks of publication). Software portability became a topic of greater import, and **DDJ** published Ron Cain's immediately famous Small-C compiler—reprinted here in full!
Contents include: The Evolution of CP/M, a CP/M-Flavored C Inerpreter, Ron Cain's C Compiler for the 8080, Further with Tiny BASIC, a Syntax-Oriented Compiler Writing Language, CP/M to UCSD Pascal File Conversion, Run-time Library for the Small-C Compiler and, as always, even more!

Vol. 6 1981

1981 saw our first all-FORTH issue (now sold out), along with continuing coverage of CP/M, small-C, telecommunications, and new languages. Dave Cortesi opened "Dr. Dobb's Clinic" in 1981, beginning one of the magazine's most popular features. Highlights: information on PCNET, the Conference Tree, and The Electric Phone Book, writing your own compiler, a systems programming language, and Tiny BASIC for the 6809.

YES! ☐ Please send me the following Volumes of **Dr. Dobb's Journal.**
☐ ALL 7 for ONLY $165, a savings of over 15%!

Payment must accompany your order.

Please charge my: ☐ Visa ☐ MasterCard ☐ American Express
I enclose ☐ Check/money order

Card #________________ Expiration Date____________

Signature____________________________

Name________________ Address________________

City________________ State________ Zip________

Vol. 1______	x	$26.75	=______
Vol. 2______	x	$27.75	=______
Vol. 3______	x	$27.75	=______
Vol. 4______	x	$27.75	=______
Vol. 5______	x	$27.75	=______
Vol. 6______	x	$27.75	=______
Vol. 7______	x	$30.75	=______
All 7______	x	$165.00	=______
Sub-total			$______

Postage & Handling Must be included with order.
Please add $1.25 per book in U.S. ($3.25 each surface mail outside U.S. Foreign Airmail rates available on request.

Mail to: Dr. Dobb's Journal, 2464 Embarcadero Way, Palo Alto, CA 94303

Code 100

Allow 6–9 weeks for delivery.

Circle **no. 82** on reader service card.

TOTAL $______

DR DOBB'S JOURNAL BRINGS RESTON BOOKS TO YOU

SCIENTIFIC PASCAL
by Harley Flanders

A refreshing, stimulating book on the most rapidly growing structured programming language. The book is filled with over 400 examples and exercises with graded levels of difficulty. The state-of-the-art programs involve approximation, zeros of functions, differentiation and integration, differential equations, matrices, characteristic roots, fast Fourier transform, permutations—anything and everything connected with scientific programming and numerical analysis.

1984 576 pages $21.95 (soft)

THE PROGRAMMER'S CP/M NOTEBOOK
by David E. Cortesi

Displays the process of designing and building software for the popular CP/M operating system. Twelve complete, useful utility programs are contained in the book. For each program the author analyzes the user's needs and develops a verbal specification, then designs the program in a high-level language, and presents the assembly language code that implements the program.

1983 225 pages $17.95 (soft)

PARALLEL PROGRAMMING IN ANSI STANDARD ADA
by George Cherry

A thorough introduction to parallel programming and algorithms in the military and civilian standard language, Ada. Uses graphical means (Petri nets) to introduce the basic concepts of concurrency, synchronization, and rendezvous. Illustrates the lanugage features with complete programs rather than program fragments. Treats task pipelines, exception handling in parallel programs, and algorithms that are optimized for the number of processors available.

1984 224 pages $21.95 (hard)

SYSTEM PROGRAMMING UNDER CP/M-80
by Lawrence E. Hughes

A thorough introduction to 8080/8085 assembly language programming. Contains a great deal of data on how to program under the CP/M operating system. Gives detailed information on installing and modifying CP/M. Includes actual usable utility programs for CP/M, along with a detailed analysis of each.

1983 208 pages $17.95 (soft)

A USER FRIENDLY GUIDE TO CP/M
by James T. Perry and Robert F. McJunkins

This guide is perfect for first-time users—taking them from an introduction to CP/M, to creating new dBASE commands. In between, the reader will find topics such as built-in commands, copying a large number of files, system files, user area, copying selected parts of files, introduction to dBASEII, altering the data base contents, and more.

1983 150 pages $16.95 (soft)

PERSONAL PASCAL: COMPILED PASCAL FOR THE IBM PERSONAL COMPUTER
by David E. Cortesi and George W. Cherry

Prepares the reader to use the Pascal programming language and compiled Pascal on the IBM Personal Computer in particular. Illustrates stepwise refinement to help the reader complete designs. Gives careful descriptions and illustrations of Pascal syntax to aid the reader in avoiding common syntactic errors. Emphasizes clarity of design and protability of code throughout.

1984 432 pages $17.95 (soft)

THE SMALL-C HANDBOOK
by James E. Hendrix

A complete description of the Small-C programming language and compiler for those who are already programming in other languages. Section 1 covers program translation concepts, presenting a survey of machine language concepts, assembly language, and the use of assembler loaders and linkers. Section 2 introduces the Small-C language, and Section 3 describes the compiler itself, covering I/O concepts, standard functions, using the compiler to generate new versions of itself, and more. In addition, several appendices offer valuable reference information on Small-C programming.

1984 272 pages $17.95 (soft)

Mail coupon to: Dr. Dobb's Journal/2464 Embarcadero Way/Palo Alto, CA 94303

Yes! Please send me the book(s) in the quantities I've indicated below:

—— Scientific Pascal, soft, R6931-1, $21.95
—— The Programmer's CP/M Notebook, soft, R5641-7, $17.95
—— Parallel Programming in ANSI Standard ADA, hard, R5434-7, $21.95
—— The Small-C Handbook, soft, R7012-9, $17.95
—— Personal Pascal: Compiled Pascal for the IBM Personal
—— Computer, soft, R5522-9, $17.95
System Programming Under CP/M-80, soft, R7456-8, $17.95
—— A User-Friendly Guide to CP/M, soft, R8117-5, $16.95

☐ Enclosed is my payment of ________ in check or money order.
☐ VISA ☐ MasterCard
Card No.: ____________
Expiration date: ____________

Name: ____________
Address: ____________
City: ____________ State: ________ Zip: ________

Please allow 4-6 weeks for delivery. Add $1.50 per book for shipping and handling charges.

Circle **no. 82** on reader service card.

Code 100

(Listing Continued, text begins on page 98)

Listing Three

```
        SHLD    BYTECNT ; SET COUNT
OKRET1:
        CALL    GETADR  ; GET ADDRESSES
        GETRG
        MVI     A,0FFH  ; SET NZ FLAG
        ORA     A       ; NO ERROR RETURN
        LDA     BYTE    ; GET BYTE
        RET
;
;   PUT NEXT BYTE INTO BUFFER/FILE
;       ON INPUT, A=CHAR AND DE PTS TO IOCB
;       ON OUTPUT, Z FLAG SET IF WRITE ERROR
;
FX$PUT::
        PUTRG           ; SAVE REGS
        STA     BYTE    ; SAVE BYTE TO OUTPUT
        CALL    PUTADR  ; PUT ADDRESSES
        LHLD    BYTECNT ; CHECK TO SEE IF ANY ROOM LEFT
        MOV     A,H
        ORA     L
        JNZ     FXPUT1
        CALL    FXOO0   ; FLUSH BUFFER AND RESTART
        JZ      ERRET   ; ERROR
FXPUT1:
        LHLD    BYTENXT ; GET PTR TO NEXT BYTE
        LDA     BYTE    ; GET NEXT BYTE
        MOV     M,A     ; STORE IT
        INX     H       ; PT TO NEXT
        SHLD    BYTENXT
        LHLD    BYTECNT ; COUNT DOWN
        DCX     H
        SHLD    BYTECNT
        JMP     OKRET1  ; OK RETURN WITH BYTE AND ADDRESSES
;
;   ROUTINE TO PUT ADDRESS BUFFERS FOR LATER USE
;
PUTADR:
        XCHG            ; PT TO BUFFER WITH HL
        SHLD    BUFFER  ; PUT BUFFER ADDRESS
        MOV     A,M     ; GET BLOCK COUNT
        STA     BCNT
        INX     H
        MOV     A,M     ; GET EOF FLAG
        STA     EOF
        INX     H
        MOV     E,M     ; GET LOW COUNT
        INX     H
        MOV     D,M
        XCHG
        SHLD    BYTECNT ; PUT BYTE COUNT
        XCHG
        INX     H
        MOV     E,M     ; GET LOW COUNT
        INX     H
        MOV     D,M
        XCHG
        SHLD    BYTENXT ; PUT NEXT BYTE PTR
        XCHG
        INX     H
        MOV     E,M     ; GET LOW COUNT
        INX     H
        MOV     D,M
        XCHG
        SHLD    BUFADR  ; PUT BUFFER ADDRESS PTR
        XCHG
        INX     H
        SHLD    FCB     ; ADDRESS OF FCB
        RET
;
;   ROUTINE TO GET ADDRESS BUFFERS BACK FOR CALLING ROUTINE
;
GETADR:
```

(Continued on page 119)

DDJ BACK ISSUES

#69 Volume VII, No. 7:
IBM-PC Issue: CP/M-86 vs. MSDOS (A Technical Comparison)—Hi-Res Graphics on the IBM-PC—PDP-1802, Part II—Review of Word Processors for IBM.

#70 Volume VII, No. 8:
Argum "C" Command Line Processor—SEND/RECEIVE File Transfer Utilities—Intel's 8087 Performance Evaluation.

#71 Volume VII, No. 9:
FORTH Issue: Floating-Point Package—H-19 Screen Editor—Relocating, Linking Loader—Z8000 Forth—Forth Programming Style—8086 ASCII-Binary Conversion Routines—CP/M Conditional SUBMIT.

#72 Volume VII, No. 10:
Portable Pidgin for Z80—68000 Cross Assembler, Part I—MODEM and RCP/Ms—Simplified 68000 Mnemonics—Nested Submits—8086/88 Trig Lookup.

#73 Volume VII, No. 11:
Wildcard UNIX Filenames—Tests for Pidgin—68000 Cross Assembler Listing, Part 2—Adding More BDOS Calls—The Perfect Hash—BASIC Memory Management—Benchmarks for CP/M-86 vs. MSDOS, and the 8087.

#76 Volume VIII, Issue 2:
PISTOL, A Forth-like Portably Implemented Stack Oriented Language—Program Linkage by Coroutines. Forth to BASIC—Linking CP/M Functions to Your High-Level Program—Concurrent CP/M-86—CP/M-80 Expansion Card for the Victor 9000—REVAS Disassembler.

#77 Volume VIII, Issue 3:
The Augusta P-Code Interpreter—A Small-C Operating System—6809 Threaded Code: Parametrization and Transfer of Control—A Common-Sense Guide to Faster, Small BASIC—A Fundamental Mistake in Compiler Design—Basic Disk I/O, Part I.

#78 Volume VIII, Issue 4:
RECLAIM Destroyed Directories—Binary Magic Numbers—8080 Fig-Forth Directory & File System—SAY" Forth Votrax Driver—TRS-80 8080 to Z80 Translator—Basic Disk I/O, Part II.

#79 Volume VIII, No. 5:
The Augusta Compiler—A Fast Circle Routine—Enhancing the C Screen Editor—Shifts and Rotations on the Z80—The SCB, TSX, and TXS Instructions of the 6502 and 6800—MS-DOS vs. CP/M-86—Controlling MBASIC—The Buffered Keyboard—IBM PC Character Set Linker—Flip Utility for the IBM PC.

#80 Volume VIII, Issue 6:
Fast Divisibility Algorithms—B-Tree ISAM Concepts—CP/M BDOS AND BIOS Calls for C—Serial Expansion in Forth—Fast Matrix Operations in Forth, Part I—Yes, You Can Trace Through BDOS—Julian Dates for Microcomputers—8088 Addressing Modes—8088 Line Generator—CP/M Plus.

#81 Volume VIII, Issue 7:
The Augusta Compiler, continued—RED: A Better Screen Editor, Part I—Anatomy of a Digital Vector and Curve Generator—Fast Matrix Operations in Forth, Part II—The AGGHHHI Program—MBOOT Revisited—CP/M Plus Feedback—MS-DOS Rebuttal—68000 Tools—Sizing Memory on the IBM PC.

#82 Volume VIII, Issue 8:
Serial-to Parallel: A Flexible Utility Box—McWORDER: A Tiny Text Editor—And Still More Fifth Generation Computers—Specialist Symbols and I/O Benchmarks for CP/M Plus—CP/M Plus Memory Management—Zero Length File Test—PAUSEIF, QUITIF, and now SKIPIF—ACTxx Cross Assemblers.

#83 Volume VIII, No. 9:
FORTH ISSUE: Forth and the Motorola 68000—Nondeterministic Control Words in Forth—A 68000 Forth Assembler—GO in Forth—Precompiled Forth Modules—Signed Integer Division—Some Forth Coding Standards—The Forth Sort.

#84 Volume VIII, No. 10:
Unix to CP/M Floppy Disk File Conversion—A Small-C Help Facility—Attaching a Winchester Hard Disk to the S-100 Bus—Using Epson Bit-Plot Graphics—8086/88 Function Macros—Auto Disk Format Selection—CP/M Plus Device Tables.

#85 Volume VIII, Issue 11:
A Kernel for the MC68000—A DML Parser—Towards a More Writable Forth Syntax—Simple Graphics for Printer—Floating-Point Benchmarks.

#86 Volume VIII, Issue 12:
Faster Circles for Apples—Cursor Control for Dumb Terminals—Dysan's Digital Diagnositic Diskette—Interfacing a Hard Disk Within a CP/M Environment—The New MS-DOS EXEC Function.

#87 Volume IX, Issue 1:
A Structured Preprocessor for MBASIC—A Simple Window Package—Forth to PC-DOS Interface—Sorted Diskette Directory Listing for the IBM PC—Emulate WordStar on TOPS-20—More on optimizing compilers—The PIP mystery device contest.

#88 Volume IX, Issue 2:
Telecommunications Issue: Micro to Mainframe Connection—Communications Protocols—Unix to Unix Network Utilities—VPC: A Virtual Personal Computer for Networks—PABX and the Personal Computer—BASIC Language Telecommunications Programming—U.S. Robotics S-100 Card Modem.

#89 Volume IX, Issue 3:
RSA: A Public Key Cryptography System, Part I—Introduction to PL/C: Programming Language for Compilers—Program Design Using Pseudocode—More on Binary Magic Numbers—How fast is CP/M Plus?—CP/M 2.2 BIOS Function: SELDSK—The results of the Floating-Point benchmark.

#90 Volume IX, Issue 4:
Optimizing Strings in C—Expert Systems and the Weather—RSA: A Public Key Cryptography System, Part II—Several items on CP/M Plus, CP/M v2.2 Compatibility—BDOS Function 10: Vastly Improved—More on MS-DOS EXEC Function—Low-Level Input-Output in C.

#91 Volume IX, Issue 5:
Introduction to Modula-2 for Pascal Programmers—Converting Fig-Forth to Forth-83—Sixth Generation Computers—A New Library for Small-C—Solutions to Quirks in dBASE II.

#92 Volume IX, Issue 6:
CP/M on the Commodore 64—dBASE II Programming Techniques—First Chinese Forth: A Double-Headed Approach—cc-A Driver for a Small-C Programming System—A New Library for Small-C (Part II)—Comments on Sixth Generation Computers—Review of Turbo Pascal.

#93 Volume IX, Issue 7:
RSX under CP/M Plus—p-A Small-C Preprocessor—A Simple Minimax Algorithm—Languages and Parentheses (A Suggestion for Forth-like Languages)—Comments on assembly language development packages, RSX to patch CP/M 2.2 with CP/M, iRMS-86 for the IBM PC, C Programming Tools.

#94 Volume IX, Issue 8:
SCISTAR: Greek and Math Symbols with WordStar—A File Comparator for CP/M Plus—Designing a File Encryption System—A Small-C Concordance Generator.

#95 Volume IX, Issue 9:
Forth Special Issue!—File Maintenance in Forth—Forth and the Fast Fourier Transform—Computing with Streams—A Forth Native-Code Cross Compiler for the MC68000—The FVG Standard Floating-Point Extension—CP/M Plus: Interbank Memory Moves Without DMA—ways to make C more powerful and flexible.

#96 Volume IX, Issue 10:
More dBASE II Programming Techniques—Simple Calculations with Complex Numbers—GREP. C: A Unix-like Generalized Regular Expression Parser—An optimization scheme for compilers, MSDOS 2.0 Filters, Sizing RAM under MSDOS, Two programming systems illustrating Runge-Kutta integration.

#97 Volume IX, Issue 11:
Adding Primitive I/O Functions to muLISP—Program Monitor Package: Using Interrupts to Instrument Applications—CP/M 2.2 Goes PUBlic—A Guide to Resources for the C Programmer—RESORT.

#98 Volume IX, Issue 12:
Varieties of Unix—Unix Device Drivers—A Unix Internals Bibliography—A file Browser Program—An Introduction to Parsing.

#99 Volume X, Issue 1:
Fatten Your Mac—QuickDraw Meets Image Writer—Archiving Files with CP/M 80 and CP/M 86—MBOOT and MODEM for the C-64's CP/M—Unstructured Forth Programming: An Introduction.

TO ORDER:

Send $3.50 per issue to: **Dr. Dobb's Journal,** 2464 Embarcadero Way, Palo, Alto, CA 94303

Please send me the issue(s) circled: **69 70 71 72**
73 76 77 78 79 80 81 82 83 84 85 86
87 88 89 90 91 92 93 94 95 96 97 98 99

I enclose $________ (U.S. check or money order).
Outside the U.S., add $.50 per issue.

I have read the postal instructions and understand that I will not receive my order unless I have sent the correct payment amount.

Please charge my: ☐ Visa ☐ M/C ☐ Amer. Exp.

Card No. ______________________ Exp. Date________

Signature ______________________________

Name ______________________________

Address ______________________________

City ______________________________

State ________________ Zip ____________

Availability on first come/first serve basis. Outside the U.S. add $.50 per issue ordered. Price includes issue, handling, and shipment by second class or foreign surface mail. Within the U.S., please allow 6–9 weeks to process your order second class. For faster service within the U.S., we'll ship UPS if you *add* $1.00 for 1–2 issues and $.50 for *each issue* thereafter. We need a street address, not a P.O. Box. Airmail rates: To Canada *add* $1.75 *per magazine*, all other foreign *add* $3.00 *per magazine*.

Circle **no. 81** on reader service card. Code 100

Eco-C Compiler
Release 3.0

NEW RELEASE

We think Rel. 3.0 of the Eco-C Compiler is the fastest full C available for the Z80 environment. Consider the evidence:

Benchmarks*
(Seconds)

Benchmark	Eco-C	Aztec	Q/C
Seive	29	33	40
Fib	75	125	99
Deref	19	CNC	31
Matmult	42	115	N/A

**Times courtesy of Dr. David Clark*
CNC - Could Not Compile
N/A - Does not support floating point

We've also expanded the library (120 functions), the user's manual and compile-time switches (including multiple non-fatal error messages). The price is still $250.00 and includes Microsoft's MACRO 80. As an option, we will supply Eco-C with the SLR Systems assembler - linker - librarian for $295.00 (up to six times faster than MACRO 80).

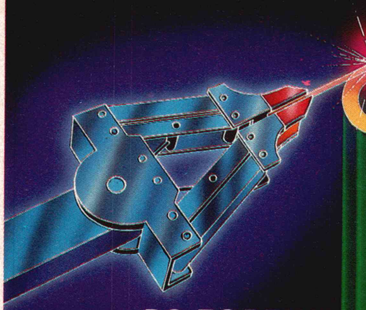
For additional information, call or write:

ECOSOFT INC. (317) 255-6476
6413 N. College Ave. • Indianapolis, Indiana 46220

VISA MasterCard

Circle **no. 35** on reader service card.

TOTAL CONTROL WITH LMI FORTH



PC FORTH™

IBM PC & XT, HP-150, Macintosh, Apple II, CompuPro, Sage & CP/M-68K, Wang PC,
All CP/M and MSDOS computers.

Try the *professional language* offering the *utmost performance* in the *shortest development time.* Transport your applications between any of our enhanced 83-Standard compilers or expanded 32-bit versions. Choose from our wide selection of programming tools including native code compilers, cross-compilers, math coprocessor support, and B-Tree file managers. All fully supported with hotline, updates, and newsletters.

Laboratory Microsystems Incorporated
Post Office Box 10430, Marina del Rey, CA 90295
Phone credit card orders to (213) 306-7412

Circle **no. 55** on reader service card.

Pascal and C Programmers

Your programs can now compile the *FirsTime*™

FirsTime is an intelligent editor that knows the rules of the language being programmed. It checks your statements as you enter them, and if it spots a mistake, it identifies it. *FirsTime* then positions the cursor over the error so you can correct it easily. *FirsTime will identify all syntax errors, undefined variables, and even statements with mismatched variable types.* In fact, any program developed with the *FirsTime* editor will compile on the first try.

More than a syntax checker!

FirsTime has many unique features found in no other editor. These powerful capabilities include a zoom command that allows you to examine the structure of your program, automatic program formatting, and block transforms.

If you wish, you can work even faster by automatically generating program structures with a single key-stroke. This feature is especially useful to those learning a new language, or to those who often switch between different languages.

Other Features: Full screen editing, horizontal scrolling, function key menus, help screens, inserts, deletes, appends, searches, and global replacing.

Programmers enjoy using *FirsTime.* It allows them to concentrate on program logic without having to worry about coding details. Debugging is reduced dramatically, and deadlines are more easily met.

FirsTime for PASCAL	$245
FirsTime for C	$295
Microsoft PASCAL Compiler	$245
Microsoft C Compiler	$395
Demonstration disk	$25

Get an extra **$100 off** the compiler when it is purchased with ***FirsTime.***
(N.J. residents please add 6% sales tax.)

Spruce
Technology Corporation
110 Whispering Pines Drive
Lincroft, N.J. 07738
(201) 741-8188 or (201) 663-0063

Dealer enquiries welcome. Custom versions for computer manufacturers and language developers are available.

FirsTime is a trademark of Spruce Technology Corporation.

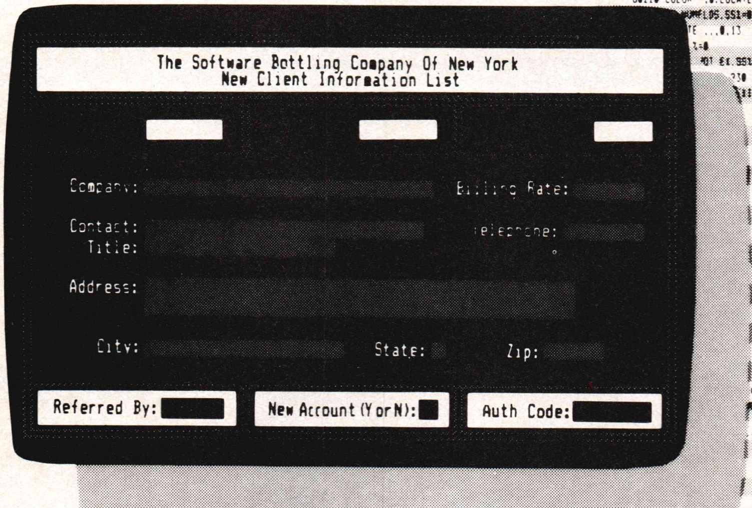
MasterCard VISA

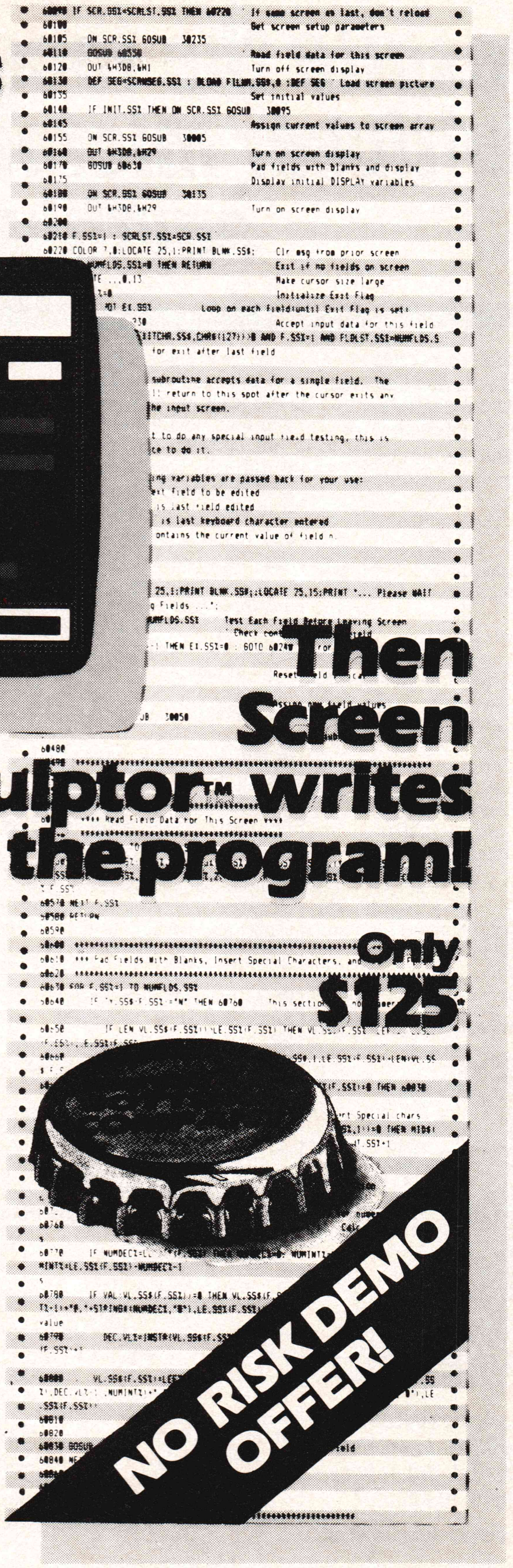
Circle **no. 65** on reader service card.

Screen Sculptor™ writes programs!

In IBM Basic
Turbo Pascal &
IBM Pascal

Create screens like this in minutes.





Then Screen Sculptor™ writes the program!

Only $125

NO RISK DEMO OFFER!

In Basic and Pascal

It's that easy! The same Screen Sculptor package generates programs in IBM Basic, Turbo Pascal, and IBM Pascal.

Now, anyone can have attractive, intelligent input screens and sophisticated data entry routines in minutes.

Move pieces of the screen around, select colors from a menu, draw lines and boxes, paint, repeat last character in any direction. And more!

Specify variable names, data types, acceptable data ranges, pictures for edit checking, etc.

Screen Sculptor then generates actual program source code based on your screen design. Use it as is or modify it.

Requires an IBM PC, XT, PCjr, PC AT or 100% compatible, 128K, DOS, one 320K disk drive and any 80 column display.

30 day no-risk demo offer

Order now and also get our free demo disk. Use the demo and the manual for up to 30 days. If you're still not convinced of Screen Sculptor's power, return the package for a full refund!

Credit card orders call 24 hrs/day, 1 (800) 824-7888, operator 268.

All other orders and inquiries call or write: Software Bottling Co, 29-14 23rd Ave, Long Island City, NY 11105, (718) 728-2200. *NYS residents add 8.25% sales tax. Item -1130

"Despite the recent press notices, multiuser microcomputers aren't anything new!"

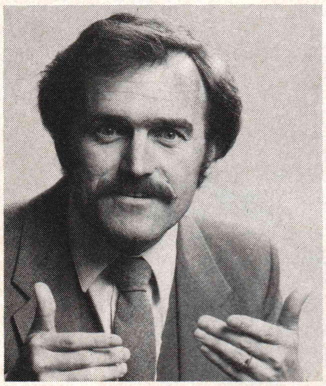
This is the first in a series of discussions with Rod Coleman, President of Stride Micro (formerly Sage Computer) on the 68000 multiuser market and its current environment.

Q: Why do you say that?

RC: "The technology to build a high performance multiuser system has been around for five years. And while some of the leaders in this industry have been pretending that micro multiuser didn't exist, we've been shipping complete systems for nearly three years. The benefits of multiuser are undeniable; it is more cost effective, and offers greater flexibility and utility. But until just recently, the marketing pressure to be compatible instead of being better, has blinded the industry."

Q: What do you mean?

RC: "Well, for example, the Motorola 68000 processor introduced 16/32-bit technology to the personal computer world a long time ago. It was fully capable of



"A surprising feature is compatibility. Everybody talks about it, but nobody does anything about it."

meeting high performance and multiuser design requirements in 1980. Instead of this trend taking off, most energy was spent promoting 8088/8086 products that were clearly inferior from a technical point of view. This phenomenon leads me to believe that they will soon rewrite the old proverb: 'Build a better mousetrap and the world will beat a path to your door,' but only if they can find the way through the marketing fog."

Q: Are things changing now?

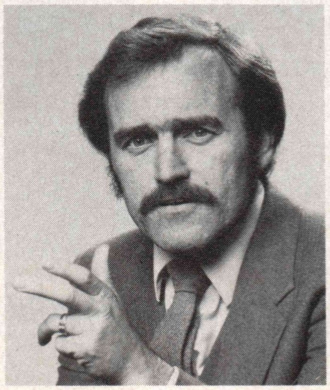
RC: "Yes and no. With the business world starting to take more and more interest in microcomputer solutions, the advantages of a solid multiuser system couldn't be kept hidden forever; companies like ours and a few others were beginning to make a dent. Instead of taking a fresh approach, some of the newest multiuser offerings will probably only give the technology an undeserved black eye! Multiuser is far more than the ability to plug in more terminals. It involves things like machine compatibility, fast processors, adequate memory, large storage capacities, backup features, networking, and operating system flexibility."

Q: Is this what makes the new Stride 400 Series different?

RC: "Exactly. That sounds self-serving, but it's true. Today a number of companies are introducing their first multiuser system. We've been building and shipping multiuser machines for almost three years. We know the pitfalls, we've fallen into some of them. But we have learned from our mistakes."

Q: Give me some examples.

RC: A hard disk is almost mandatory for any large multiuser installation. Yet, backing up a hard disk can be a nightmare if you only have floppies to work with. That's why we've added a tape backup option to all the larger Stride 400 Series machines. It's irresponsible for a manufacturer to market a multiuser system without such backup. Another good lesson was bus design. We started with one of our own designs, but learned that it's important not only to find a bus that is powerful, but also one that has good support and a strong future to serve tomorrow's needs. We



"The marketing pressure to be compatible instead of being better, has blinded the industry."

think the VMEbus is the only design that meets both criteria and thus have made it a standard feature of every Stride 400 Series machine."

Q: What are some of the other unique features of the 400 Series?

RC: "A surprising feature is compatibility. Everybody talks about it, but nobody does anything about it. Our systems are completely compatible with each other from the 420 model starting at $2900, through the 440, on to the powerful 460 which tops out near $60,000. Each system can talk to the others via the standard built-in local area network. Go ahead and compare this with others in the industry. You'll find their little machines don't talk to their big ones, or that the networking and multiuser are incompatible, or that they have different processors or operating systems, and so on."

Q: When you were still known as Sage Computer, you had a reputation for performance, is that still the case with the new Stride 400 Series?

RC: "Certainly, that's our calling card: 'Performance By Design.' Our new systems are actually faster; our standard processor is a 10 MHz 68000 running with no wait states. That gives us a 25% increase over the Sage models. And, we have a 12 MHz processor as an option. Let me add that speed isn't the only way to judge performance. I think it is also measured in our flexibility. We support a dozen different operating systems, not just one. And our systems service a wide variety of applications from the garage software developer to the corporate consumer running high volume business applications."

Q: Isn't that the same thing all manufacturers say in their ads?

RC: "Sure it is. But to use another over used-term, 'shop around'. We like to think of our systems as 'full service 68000 supermicrocomputers.' Take a look at everyone else's literature and then compare. When you examine cost, performance, flexibility, and utility, we don't think there's anyone else in the race. Maybe that's why we've shipped and installed more multiuser 68000 systems than anyone else."

STRIDE MICRO

Formerly Sage Computer

For more information on Stride or the location of the nearest Stride Dealer call or write us today. We'll also send you a free copy of our 32 page product catalog.

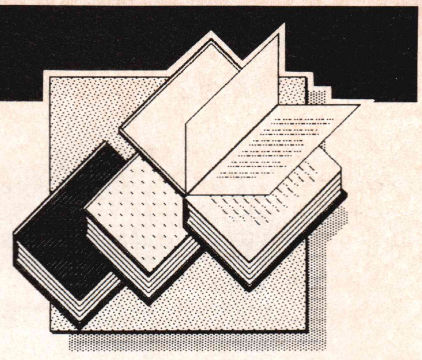
Corporate Offices:
4905 Energy Way
Reno, NV 89502
(702) 322-6868

Regional Offices:
Boston: (617) 229-6868
Dallas: (214) 392-7070

Listing Three

```
        LHLD    BUFFER  ; PT TO BUFFER ADDRESS
        INX     H       ; SKIP BLOCK COUNT
        LDA     EOF     ; SET EOF FLAG
        MOV     M,A
        XCHG
        LHLD    BYTECNT ; SET BYTE COUNT
        XCHG
        INX     H
        MOV     M,E     ; PUT LOW COUNT
        INX     H
        MOV     M,D
        XCHG
        LHLD    BYTENXT ; SET NEXT BYTE PTR
        XCHG
        INX     H
        MOV     M,E     ; PUT LOW COUNT
        INX     H
        MOV     M,D
        RET
;
;   BUFFERS
;
BYTE:           DS      1       ; INPUT BYTE
BUFFER:         DS      2       ; STARTING ADDRESS OF I/O CONTROL BLOCK
;
;   THE FOLLOWING MIRRORS THE STRUCTURE OF THE I/O CONTROL BLOCK
;
BCNT:           DS      1       ; NUMBER OF BLOCKS
EOF:            DS      1       ; EOF FLAG (0=NOT AT EOF, 0FFH=AT EOF)
BYTECNT:        DS      2       ; NUMBER OF BYTES TO GO YET
BYTENXT:        DS      2       ; ADDRESS OF NEXT BYTE TO PUT/GET
BUFADR:         DS      2       ; ADDRESS OF WORKING BUFFER
FCB:            DS      2       ; ADDRESS OF FCB

        END
```

End Listings



Diana, An Intermediate Language for Ada

Edited by G. Goos, W. A. Wulf, A. Evans Jr., and K. J. Butler
Published by Springer-Verlag
$11.00, 201 pages
Reviewed by John R. Johnson

Most computer language compilers in recent years have used a concept called "common back end" in their implementation. We normally think of a compiler as a program that accepts a file of statements in some language as input and provides as output a file of machine language codes that will implement the intent of the input file on a specific computer. Many compilers were created in just this way. However, significant advantages can be gained by writing compilers for higher-level languages to provide their output in a more generic lower-level language for easy implementation on a variety of machines. The two major advantages to this approach are portability and flexibility.

Portability can be gained by using a low-level language as an intermediate or "common back end" language. My high-level compiler itself, which represents a serious programming effort, can be written in a high-level language or even in the intermediate language it produces. This simple intermediate language then can be implemented on a variety of machines to create a family of high-level compilers for many different computers. A large part of the work of high-level language implementation has to be done only once. A good example of this approach is the p-code intermediate language used in a number of implementations of the Pascal language.

Flexibility is gained by using the same intermediate language for a number of different high-level language compilers. It becomes a relatively easy task to combine modules or routines written in different source languages into one program. A good example of this is the Microsoft family of high-level language compilers: they all compile to a common relocatable assembly language, and the modules are linked with a common linker/loader to produce executable code.

To make use of this powerful technique, you need an intermediate language that is both easily implemented and capable of supporting all of the constructs of the high-level language. Because Ada contains a number of features in its specification that are straight from computer science research projects, a suitable intermediate language for Ada did not exist.

Diana, An Intermediate Language for Ada not only documents such a language, but it provides an intermediate language usable for virtually any modern high-level language such as Modula II or Pascal. The book is intended as both an introduction to Diana and a language reference manual for Diana.

This book is part of a larger series, "Lecture Notes in Computer Science," published by Springer-Verlag. The books in this series that I have seen are intended primarily for graduate students in computer science. This book is not an exception.

It is a clear and concise definition and description of a language that is a powerful tool in modern high-level compiler construction. That the book is obviously written for experienced compiler writers is not a criticism: by its nature, it would be of only limited usefulness to anyone not currently engaged in designing a high-level language compiler.

The book is well conceived and presented, and I would suggest that anyone considering writing a compiler for the Ada language buy it. It should also be in the library of any modern systems programmer. If your programming interests do not extend to compiler design, however, *Diana* would most likely fail to repay the considerable effort required to read and understand it.

Ada, An Advanced Introduction Including Reference Manual for the Ada Programming Language

by Narain Gehani
Published by Prentice-Hall, Inc.
$19.95, 291 pages
Reviewed by John R. Johnson

The Ada programming language has stirred up a great deal of interest and controversy in recent years. As a result, many books have appeared on the market to expand upon Ada. This is the best one I have seen.

The book is written for the experienced programmer, preferably someone who has had some exposure to one of the ALGOL-derived languages such as Pascal or PL/I. Gehani assumes the reader is familiar with standard programming concepts such as type definitions and control structures.

The first portion of the book is devoted to a thorough but brief description of the correct Ada syntax and form for all programming constructs not unique to Ada. Gehani covers the Ada character set, type and function declarations, and subtypes with ranges and limits. The definitions are written in the concise format normally used for language specification and documentation. Any experienced programmer should have no difficulty following the material. Gehani has illustrated each situation with several examples of correct Ada code.

He then spends the bulk of the book thoroughly discussing several features of Ada that are shared with few, if any, common languages. There is an excellent discussion of the Ada "package"

concept for information hiding and data encapsulation; this is the basic facility or modularity in Ada programs. The contents of any "package" may be altered at will as long as the imports and exports from the package remain consistent with the package specification. Ada provides for separate compilation of packages. Gehani uses many examples to illustrate the specific syntax associated with the concept.

One of Ada's unusual features is the provision for concurrency of portions of a program. The author has given us a clear and concise look at the problems of concurrency. Ada has a number of features to provide coroutines, concurrent actions, and data passing between independent concurrent tasks and procedures within a program. Once again we have numerous examples to show how to handle handshaking between asynchronous independent processes using correct Ada syntax.

Ada provides a unique and complete system for exception handling within programs. Specific exception handling routines may be added at the end of any block to process any exceptions that may arise anywhere in the block. When an exception occurs, control is transferred to the exception handler. When the exception handler is finished, the block where the exception handler resides is terminated and control passes back to the enclosing block. If the exception handler is omitted, control passes back to the exception handler in the enclosing block. This concept is presented clearly with some excellent examples.

The generic facilities of Ada are an interesting variant of the standard function library concept used in C and other separate-module compilation languages. A generic package may be "instantiated" by a declaration in a new package with appropriate parameters; the result is a unique and specific application of a generic package, such as a sort routine. This unusual approach is presented with some useful examples.

The balance of the first part of the book is devoted to a discussion of the separate compilation features and those features that may be implementation-dependent. Here, also, Gehani has used precise examples.

For all of the examples in the book, the author has used stepwise refinement to work from a general problem statement to a set of specific routines that implement the solution in Ada. It is interesting to follow the refinement process: all of the steps are included. The book would be worth buying just to trace this stepwise refinement process.

The second half of the book is the complete reference manual for the Ada programming language—reprinted from the ANSI/MIL-STD-1815 A. This is the current official definition of the Ada language. Considerably less readable than the first part of the book, it is indispensable for anyone who wishes to attempt an implementation of the Ada language.

The book includes a comprehensive bibliography for anyone interested in additional reading on the topics covered. The two halves of the book are indexed separately for reference purposes. The indexing is well done.

This book will quickly become dog-eared on the desk of any programmer working in the Ada language. When we are forced to begin using Ada in our shop, I am going to insist that every programmer be provided with his or her own copy. It is an essential tool for the new or experienced Ada programmer.

New Books

Hackers

Steven Levy
Anchor Press/Doubleday, Garden City, New York, 1984
$17.95, 458 pages
ISBN 0-385-19195-2

Steven Levy has taken on the exegesis of the Hacker Ethic. In breathless, Right Stuff prose, he follows the followers of the hacking dream from the basements of MIT through the garages of Silicon Valley right up the slopes of the Sierra, where game programmers are treated like rock stars. Steven, did you retain the film rights?

How to Copyright Software

M.J. Salone
Nolo Press, Berkeley, California, 1984
$21.95, 256 pages
ISBN 0-917316-7

It is perhaps only to be expected that a 256-page book explaining how to copyright software would cost more than twice as much as a 475-page book telling how to get it free.

The Free Software Catalog and Directory

Robert A. Froehlich
Crown Publishers, Inc., New York, New York, 1984
$9.95, 475 pages
ISBN 0-517-55448-8

Cross-referenced directories for the CPMUG and ACJ-NJ's SIG/M public domain software libraries, plus lists of bulletin boards and users' groups throughout the U.S.

Digital Deli

Steve Ditlea, Ed.
Workman Publishing Co., New York, New York, 1984
$12.95, 382 pages
ISBN 0-89480-591-6

No way to summarize this book; it contains a short piece (all the pieces in this book are short) on how to use a word processor, a quarter-page PR photo of IBM CEO John Opel, arguments for the Eniac being an Aquarius and for the Mac being short skis for the mind, cartoons, computer art, quizzes, and memoirs. The title is not original, but it fits.

The Netweaver's Sourcebook

Dean Gengle
Addison-Wesley Publishing Company, Menlo Park, California, 1984
$14.95, 326 pages
ISBN 0-201-05208-3

How refreshing to hear that "it's still perfectly socially acceptable to hate machines arbitrarily and capriciously, *especially* answering machines." This book takes a remarkably elevated perspective on networks and communications.

The Whole Earth Software Catalog

Stewart Brand, Ed.
Quantum Press/Doubleday, Garden City, New York, 1984
$17.50, 208 pages
ISBN 0-385-19166-9

The Brand hand is as evident here as in Whole Earth non-software catalogs, and admirers of wit, irreverence, controversy, and—um, wholearthness will be glad of that. Dr. Dobb contributed to this collection of recommended programs.

Up and Running

Charles Sherman
Ashton-Tate, Culver City, California, 1984
$15.95, 312 pages
ISBN 0-912677-14-7

A collection of raw interviews with software magnates Kildall, Gates, Kapor, and others, plus David Cole on "Guerilla Management." Sherman doesn't concentrate just on conventional marketeers; there are interviews here with shareware entrepreneurs, too. No index, unfortunately.

DDJ

Save millions of dollars with Six-Shooters™

Last year 430 million business slides were made at a cost of $3.2 billion. Most of these slides were manually generated.*

These slides could have been made on Sweet-P® Six-Shooter Personal Plotters™. Faster and better. With savings of millions of $!

Save Money and Manage Better.

Use your office computer and Six-Shooter Personal Plotter to create and plot finished charts in 6 colors in 5 to 15 minutes. Save $5.00 to $100.00 per chart.

Save more money. Use your charts to:

- Reduce meeting times 28%**
- Get fast favorable decisions**
- Get your report read. "One Sweet-P picture is worth a thousand print-outs".

Don't settle for old-fashioned, slow plotters. With office costs running $10.00 to $20.00/hr., Six-Shooter performance saves a bundle. Best of all, Six-Shooter performance and quality costs less—up to 45% less than other plotters in its class

Sweet-P is a high quality American made precision machine. It's fast. It plots 14 inches per second. It's beautiful for office and technical work. Plot perfect A-size slides for business presentations. Or big B-size block diagrams. Every office, every Quality and Production Manager and every Engineer should have one.

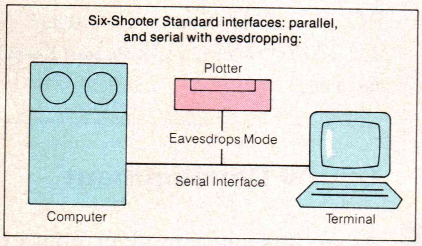
Over 100 graphics software packages drive the Six-Shooter—world famous packages like Lotus 1-2-3™ and ISSCO™, Tel-A-Graf™ and Disspla™.

The Six-Shooter holds six pens. Pens are changed automatically. Pens are capped automatically when not in use, so that pens last longer and start quicker.

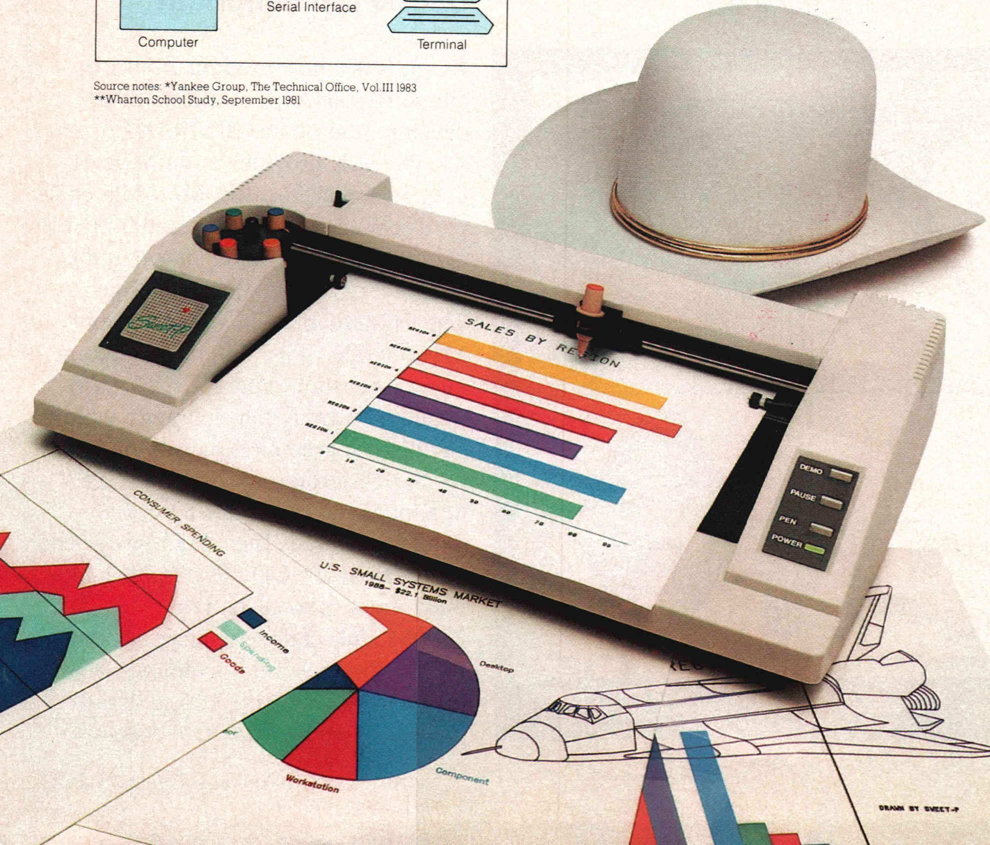
The Six-Shooter easily connects to almost any computer. It has RS-232 serial and Centronics parallel connectors. And it supports two standard graphics languages—Sweet-P Graphics Language (SPGL™) and Hewlett-Packard Graphics Language (HPGL™).

The Six-Shooter plots on almost any media. Make brilliant overhead transparencies. Plot on film, and on plain and coated papers.

Save on wiring costs too. The Six-Shooter will "eavesdrop" on the RS-232 cables that connect your terminals now. (This makes it easy for Six-Shooters to join local and long distance networks.)



Source notes: *Yankee Group, The Technical Office, Vol.III 1983
**Wharton School Study, September 1981



What about support? Six-Shooter customers get fast professional help with software, hardware and interface questions. And warranty and service support is quick. If we ever have to fix your plotter, we'll repair it in less than a week (usually 2 or 3 days).

Sweet-P®

only $1,095
(prices subject to change)

Our toll-free telephone numbers are: 800/227-4375, in California call: 800/227-4371, Telex: 181740

Enter Computer Inc.
6867 Nancy Ridge Drive
San Diego, CA 92121

Sweet-P, Six-Shooter, Personal Plotter and SPGL are trademarks of Enter Computer, Inc. Lotus 1-2-3 is a trademark of Lotus Development, Inc. Tel-A-Graf, Disspla and ISSCO are trademarks of Integrated Software System Corporation, HPGL is a trademark of Hewlett-Packard, Inc.

Circle **no. 24** on reader service card.

OF INTEREST

by R.P. Sutherland

Disk Memory

The notion of having the Library of Congress in the palm of your hand is coming closer to reality every day. A 4¾-inch compact disk ROM with **550 megabytes of storage** capacity on one side has been developed by Nippon Columbia of Kawasaki, Japan. Contact Robert Heiblim, Denon America, Inc., 27 Law Drive, Fairfield, NJ 07006 (201) 575-7810 **Reader Service No. 111.**

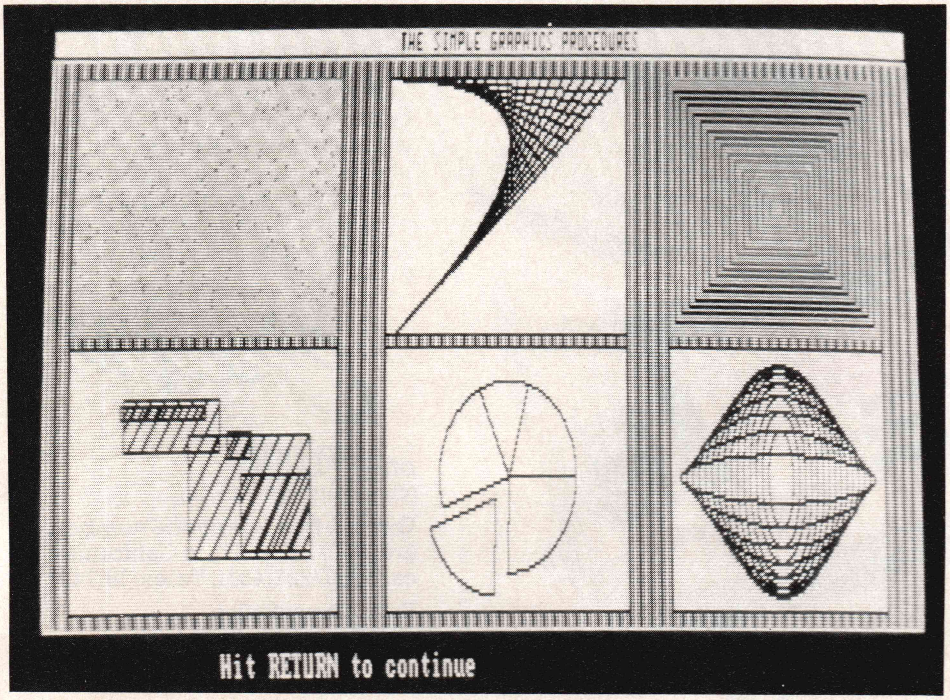
Hewlett-Packard has entered the OEM market with a 3½-inch 10 megabyte disk drive. The **HP 97501A** 3½-inch micro-Winchester is priced at $400.00 each in quantities of 10,000. For more information, write: Inquiries Manager, Hewlett-Packard Company, 1020 N.E. Circle Boulevard, Corvallis, OR 97330 **Reader Service No. 113.**

A diagnostic diskette for the IBM PC, Apple II, TRS-80, and Commodore 64 is available from Dymek Corporation. The disk is called **RID** (Recording Interchange Diagnostic), and it performs seven tests on disk drives: drive speed, radial position, hysteresis, write function, erase crosstalk, minimum signal-to-noise, and clamping. The disk is easy to use, costs $34.95, and is available from Dymek Corporation, 1851 Zanker Road, San Jose, CA 95112 (408) 947-8700 **Reader Service No. 115.**

Utilities

A high-resolution graphics system for Turbo Pascal has been released by Borland International. **Turbo Graphix Toolbox** provides procedures to create the contents of windows and to allow copying from window to window as well as scrolling of windows horizontally and vertically. There are also procedures that provide RAM storage of screen images and so permit animation in real time application, up to 500 images per second. (See photo below.) Turbo Graphix Toolbox comes with complete commented source code on disk for $49.95 from Borland International, 4113 Scotts Valley Drive, Scotts Valley, CA 95066 (408) 438-8400 **Reader Service No. 123.**

Stellation Two has disk buffering software for the IBM PC. **Invisible Optimizer** loads automatically in the unused memory space of any expansion board for the IBM PC. It works by keeping track of all disk activity and copying the most needed data into system memory. Applications then obtain the most needed data from RAM instead of from the disk drive. Invisible Optimizer is available for $69.00 from Stellation Two Inc., 26 W. Mission St., P.O. Box 2342, Santa Barbara, CA 93120 (805) 569-3132 **Reader Service No. 125.**



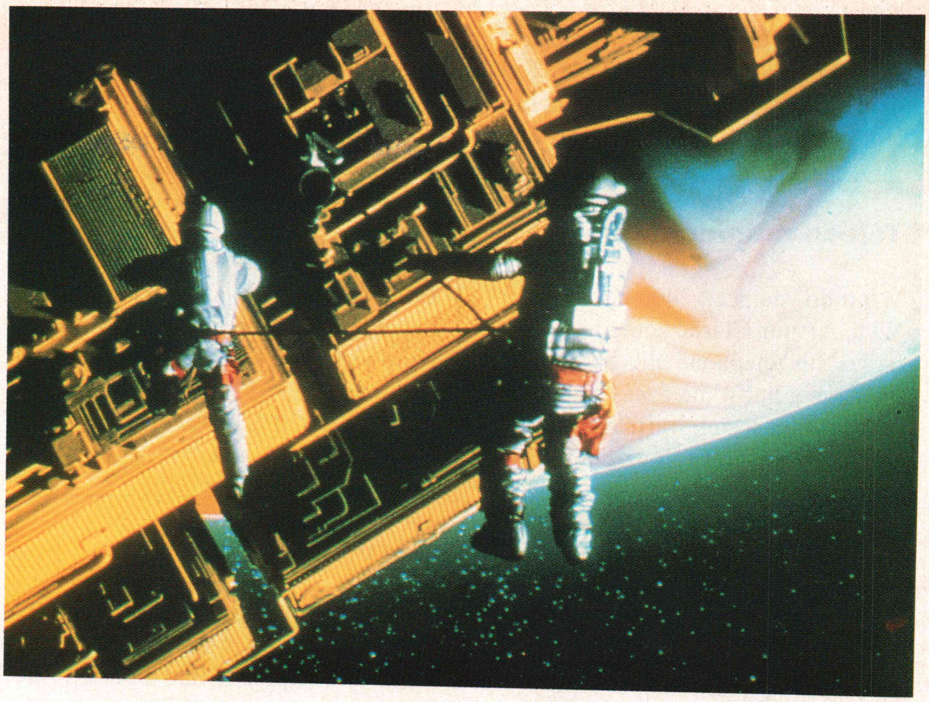
68000 Development

PC-68K is a new product from Language Resources that allows 68000 development on an IBM PC-XT or PC-AT. PC-68K provides a symbolic debugger, linker/locator, Motorola compatible macro assembler, and an IEEE floating point package. Pascal and C compilers are available as options. The PC-68K plug-in board has an 8 MHz 68000 CPU, a memory management subsystem, and 256K of RAM for 68000 family development, which can also be used as expansion memory by PCDOS programs. The basic PC-68K package costs $2995.00. Language Resources Inc., 4885 Riverbend Road, Boulder, CO 80301 (303) 449-8087 **Reader Service No. 119.**

A **Modula-2** development system for the Pinnacle microcomputer has been introduced. Interesting details include a Modula-2 native code com-

piler and a 12 MHz 68000. The package is available for $3995.00 from Pinnacle Systems, 10410 Markison Road, Dallas, TX 75238 (214) 340-4941 **Reader Service No. 121.**

2010



Arthur C. Clarke's *2001: A Space Odyssey* was written, filmed, and published before 1969, when Neil Armstrong took that giant leap. In the preface to ***2010: Odyssey Two***, in reference to *2001*, Clarke mentions "uncanny instances of nature imitating art." In *2001*, the novel, the spaceship Discovery "slingshots" its way by Jupiter to reach the Saturnian moons. The Voyager space probes in 1979 used the same technique. When Stanley Kubrick set the third confrontation between Man and Monolith among the moons of Jupiter, Io, Europa, Ganymede, and Callisto were just points of light in our telescopes. The discoveries of the Voyagers have since allowed us to gaze upon the surfaces of these fantastic moons. The astronauts of Apollo 8 had already seen the film when, in 1968, they became the first men to see the other side of the moon. They were tempted to report the discovery of a large black monolith! More recently, the crew of Skylab televised their discovery that they could run around the interior in much the same way in which Frank Poole, in the film, runs around the circular track of the giant centrifuge.

Arthur C. Clarke, one of the world's best known and most respected authors of science fiction and science fact literature, poses before one of the Kaypro II computers used to relay information between the production office and Clarke's Sri Lanka home.

Pasadena's Jet Propulsion Laboratory (JPL) claims that the vision of Jupiter in the film *2010* is better than its own. The visual effects supervisor fed the raw JPL data from the Voyager fly-by mission into a Cray. The Voyager data, processed and correlated with information about cloud vortexes, produced a detailed moving image of Jupiter.

I look forward to viewing the resurrection of HAL. (By the way, in the novel there is a character who flatly

denies the correspondence of HAL to IBM.) One Hitchcockian detail: Arthur Clarke appears as a pigeon-feeding wino on a park bench outside the White House.

Telecommunications

What do you need 2400 baud for? Well, Arthur Clarke could have halved his telephone bills across thirteen and a half time zones when he sent his film script drafts from Sri Lanka to his screenplay writer in the United States. Both Hayes and U.S. Robotics have introduced **2400 baud modems**. The Hayes costs $899.00 and the U.S. Robotics costs $895.00. Hayes is at 5923 Peachtree Industrial Blvd., Norcross, GA 30092 **Reader Service No. 101**, and U.S. Robotics is at 1123 W. Washington Blvd., Chicago, IL 60607 **Reader Service No. 103.** Where do we go from here—9600 baud?

If you can't justify $900.00 for a 2400 baud modem, consider what $200.00 will do. Business Computer Network is offering a **1200 baud modem** with SuperScout software for $200.00. Business Computer Network, Inc., is located at 1000 College View Drive, Riverton, WY 82501 (800) 446-6255 **Reader Service No. 105.** In the same spirit, Artisoft Inc. has introduced communications software for $49.95. **Envoy** is a menu-driven telecommunications program with autodial capabilities and XMODEM transfer protocols. Contact Artisoft, Inc., P.O. Box 41436, Tucson, AZ 85717 (602) 327-4305 **Reader Service No. 107.**

A newly patented device called **Shuttle Communicator** allows users to download software from an AM or FM station with a radio interfaced to the computer! Shuttle Communicator costs $70.00. (See photo below.) For additional information, contact Robert Hardwick, The Microperipheral Corporation, 2565 152nd Ave. N.E., Redmond, WA 98052 (206) 881-7544 **Reader Service No. 109.**

Miscellany

Stars and Planets on the Apple II

The Observatory is a program that turns your Apple II into an electronic celestial sphere containing stars, star clusters, galaxies, and nebulae. Halley's Comet and all of the planets are included. Users have available nine levels of magnification. The author, Gary Lassiter, claims that the program is accurate for any place on the earth for any moment of time in a 10,000 year span. The Observatory sells for $125.00 from Lightspeed Software, 2124 Kittredge, Suite 185, Berkeley, CA 94704 (415) 486-1165 **Reader Service No. 127.**

Macintosh Disk Backup

COPY II MAC makes backups of protected Macintosh software. The package includes utilities to lock/unlock, protect/unprotect, and make files visible/invisible. COPY II MAC is available for $39.95 from Central Point Software, Inc., 9700 SW Capitol Highway, Suite 100, Portland, OR 97219 (503) 244-5782 **Reader Service No. 129.**

Micromouse Contest

For those who think that they can build a small self-contained robot able to navigate quickly a complicated maze, a contest will be held by the IEEE Computer Society at the Microprocessor Forum. The Microprocessor Forum will be held March 31 to April 4, 1985, at Bally's Park Place Hotel, Atlantic City, NJ. The Micromouse Contest aims to find a robot able to negotiate a maze in the shortest period of time. The robot cannot use an energy source employing a combustion engine, cannot leave part of its body behind while running the maze, cannot jump over, climb, scratch, damage, or destroy the walls that constitute the maze, and cannot be longer or wider than 25 cm. It also cannot be controlled by radio or wire. A microcomputer must be incorporated into the design to control the sensors and drive motors, to memorize the progress of the mouse through the maze, and to calculate the shortest path to the destination. For more information about the contest, write to: Micromouse Contest, IEEE Computer Society, P.O. Box 639, Silver Spring, MD 20901.

Micro Cornucopia Gets Sol Libes

This month, Sol Libes, founder of *Microsystems*, will begin a regular column in *Micro Cornucopia*. The column will cover the latest public domain software releases. *Micro Cornucopia* is the "Single Board Systems Journal" that supports the Kaypro, the Xerox 820, and the Big Boards. *Micro Cornucopia* is located at P.O. Box 223, Bend, OR 97709 **Reader Service No. 131.**

Printer Pedestal

Zavie Enterprises has a desktop printer stand of a useful design. The **Printer Pedestal** comes in dark brown and is made of ¼-inch metal rods arc-

welded together. It can support over a hundred pounds. The 80 column width costs $19.95 and the 132 column width costs $24.95. Contact Zavie Sokoloff, 484 Lakepark Avenue, Suite 186, Oakland, CA 94610 (415) 531-0302 **Reader Service No. 133.**

BASIC

Morgan Computing Company has released version two of **Professional BASIC** and lowered the price from $345.00 to $99.00. Professional BASIC requires an IBM PC or AT. The new version includes windows into program execution and provides memory access of 640K. A $49.00 enhancement package provides 8087 and 80287 support. Contact Morgan Computing Company at 10400 N. Central Expressway, Suite 210, Dallas, TX 75231 (214) 739-5895 **Reader Service No. 117.**

DDJ

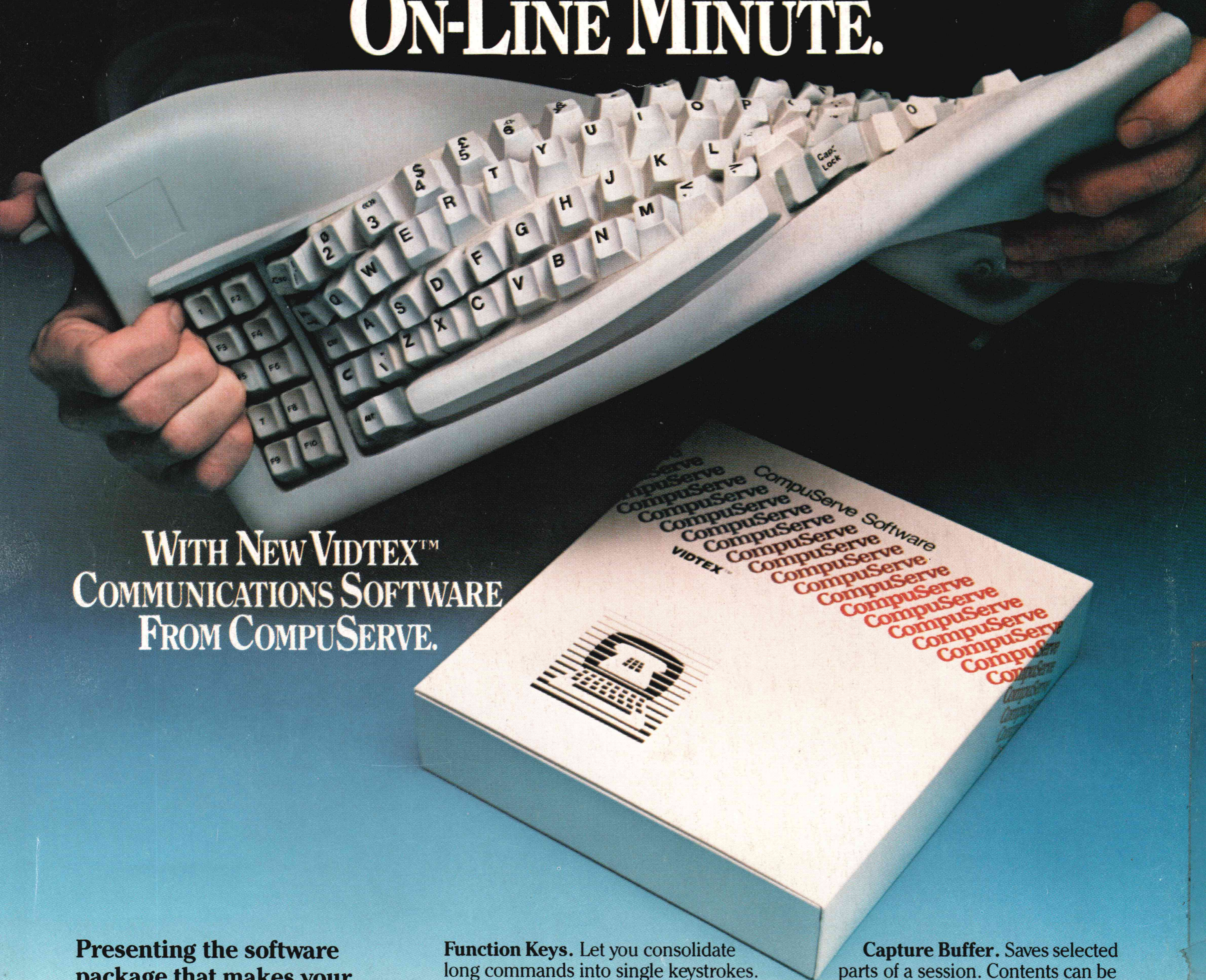
Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service No. 198.

ADVERTISER INDEX

Reader Service No.	Advertiser	Page No.
104	Action Computer Enterprises	IBC
1	Alpha Computer Service	91
5	Amanuensis, Inc.	79
3	American Planning Corporation	67
7	Ampro Computers	87
8	Apropos Technology	53
11	Automata Design Associates	109
10	Avocet Systems, Inc.	39
13	B.G. Micro	41
4	BCN	67
12	BD Software, Inc.	122
14	Borland International	C-2
21	Brady Company	81
15	C Source	97
16	C Systems	101
17	C User's Group	90
18	C Ware	59
20	Catspaw	119
6	Central Point Software	113
23	CompuServe	BC
9	Compucraft	106
32	Computer Friends	43
22	Computer Helper Industries, Inc.	87
27	Creative Solutions	31
28	D&W Digital	19
29	Data Access Corporation	2
30	Data Base Decisions	47
31	Datalight	14
52	Digital Pathways, Inc.	83
33	Digital Research Computers	45
34	Disks Plus	119
35	Ecosoft, Inc.	115
*	Edward Ream	107
24	Enter Computer	123
36	Essential Software	53
37	Faircom	103
42	Farbware	119
25	Flagstaff Engineering	85
40	Fox Software Inc.	23
38	Gimpel Software	90
43	Greenleaf Software, Inc.	27
26	Hallock Systems Consultants	128
44	Harvard Softworks	78
46	IQ Software	21
48	Illyes Systems	103
91	InfoPro Systems	91
*	Integral Quality	47
50	Interface Technologies	94-95
	Kimrich Computer Design	89
54	Korsmeyer Electronics Design	103
55	Laboratory Microsystems	115
57	Lark Software	106
58	Lattice, Inc.	101
59	Leo Electronics, Inc.	89
74	Lifeboat Associates	113

Reader Service No.	Advertiser	Page No.
84	Megamax, Inc.	105
39	MicroDynamics Corp.	89
62	MicroMotion	53
41	MicroSmith Computer Tech.	112
96	Micron Technology	79
64	Microprocessors Unlimited	109
67	Microtec Research, Inc.	13
66	Mitek	101
51	Morgan Computing Co.	109
56	Nantucket	3
60	Nicolet Paratronics	59
61	Northwest Computer Algorithms	67
68	OPT-Tech Data Processing	89
70	Phoenix Computer Products	11
71	Poor Persons Software	79
73	Procode International	63
76	QCAD Systems, Inc.	107
79	Rational Systems, Inc.	92
80	Revasco	87
85	SemiDisk Systems	105
86	Shaw American Technologies	127
87	Simpliway Products Company	105
63	Soft Advances	59
88	Softaid, Inc.	112
89	Softfocus	11
90	Software Horizons, Inc.	63
69	Software Information Systems	29
91	Software Toolworks	77
92	Softway, Inc.	121
75	Solution Systems	37
94	Solution Systems	37
95	Solution Systems	37
93	Solution Systems	97
97	Speedware	87
65	Spruce Technologies Corp.	115
83	Stride Micro	118
98	Summit Software	25
106	The Code Works	14
67	The Programmer's Shop	63
110	The Software Bottling Company	116-117
102	Thunder Software	128
77	UniPress Software	7
78	Victory Enterprises Tech.	14
112	Wendin, Inc.	9
118	Western Wares	112
114	Whitesmiths, Ltd.	1
116	Wizard Systems	127
120	DDJ Classified Ad Announcement	91
72	DDJ Advertising	103
100	DDJ Subscription	107
81	DDJ Back Issues	108, 114
82	DDJ Bound Volume/Reston	110-111
83	DDJ C Compiler	113
122	DDJ Subscription Problems	113
82	DDJ Bound Volume	104

SQUEEZE MORE OUT OF EVERY ON-LINE MINUTE.



WITH NEW VIDTEX™ COMMUNICATIONS SOFTWARE FROM COMPUSERVE.

Presenting the software package that makes your computer more productive and cost-efficient.

CompuServe's new Vidtex™ is compatible with many personal computers sold today (including Apple,® Commodore,® and Tandy/Radio Shack® brands). And it offers the following features*–and more–to let you communicate more economically with most time-sharing services (including CompuServe's Information Service).

Auto-Logon. Lets you log on to a host simply and quickly by utilizing prompts and responses defined by you. Also allows quick transmission of predefined responses to host application programs after logging on.

Function Keys. Let you consolidate long commands into single keystrokes. Definitions can be saved to and loaded from disk file, allowing multiple definitions for multiple applications.

Error-Free Uploading and Downloading. CompuServe "B" Protocol contained in Vidtex lets you transfer from your computer to CompuServe and from CompuServe to your computer anywhere in the country. Also provides error-free downloading from CompuServe's extensive software libraries.

Full Printer Support. Printer buffer automatically buffers characters until printer can process; automatically stops on-line transmission when full; and automatically resumes transmission when capacity is re-established. Also, lets you print contents of textual video screen or RAM buffer at any time.

Capture Buffer. Saves selected parts of a session. Contents can be written to a disk file; displayed both on and off line; loaded from disk; and transmitted to the host.

On-line Graphics. Integral graphics protocol displays stock charts, weather maps and more.

If you are already a CompuServe subscriber, you can order Vidtex on line by using the GO ORDER command. Otherwise, check with your nearest computer dealer; or to order direct, call or write:

CompuServe

P.O. Box 20212, 5000 Arlington Centre Blvd.
Columbus, Ohio 43220

1-800-848-8199
In Ohio, call 614-457-0802

An H&R Block Company

*Some versions of the Vidtex software do not implement all features listed.

Circle **no. 23** on reader service card.

Vidtex is a trademark of CompuServe, Incorporated. Apple is a trademark of Apple Computer, Inc. Commodore is a trademark of Commodore Business Machines. Radio Shack is a trademark of Tandy Corp.